

Oracle® Identity Management

Application Developer's Guide

10g Release 2 (10.1.2)

B14087-02

July 2005

B14087-02

Copyright © 1999, 2005, Oracle. All rights reserved.

Primary Author: Ellen Desmond

Contributing Author: Richard Smith

Contributors: Vasuki Ashok, William Bathurst, Tridip Bhattacharya, Kamalendu Biswas, Ramakrishna Bollu, Saheli Dey, Bruce Ernst, Rajinder Gupta, Ganesh Kirti, Ashish Kolli, Stephen Lee, David Lin, Radhika Moolky, Nithya Mulalidharan, Samit Roy, David Saslav, David Su

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software—Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Retek are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Portions of this document are from "The C LDAP Application Program Interface," an Internet Draft of the Internet Engineering Task Force (Copyright (C) The Internet Society (1997-1999). All Rights Reserved), which expires on 8 April 2000. These portions are used in accordance with the following IETF directives: "This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English."



RSA and RC4 are trademarks of RSA Data Security. Portions of Oracle Internet Directory have been licensed by Oracle Corporation from RSA Data Security.

Oracle Directory Manager requires the Java™ Runtime Environment. The Java™ Runtime Environment, Version JRE 1.1.6. ("The Software") is developed by Sun Microsystems, Inc. 2550 Garcia Avenue, Mountain View, California 94043. Copyright (c) 1997 Sun Microsystems, Inc.

This product contains SSLPlus Integration Suite™ version 1.2, from Consensus Development Corporation.
iPlanet is a registered trademark of Sun Microsystems, Inc.

Contents

Preface	xxi
Audience	xxi
Documentation Accessibility	xxi
Related Documents	xxii
Conventions	xxiii
What's New in the SDK?	xxv
New Features in the Release 10.1.2 SDK	xxv
New Features in the Release 9.0.4 SDK	xxv
Part I Programming for Oracle Identity Management	
1 Developing Applications for Oracle Identity Management	
Benefits of Integrating with Oracle Identity Management	1-1
Oracle Identity Management Services Available for Application Integration	1-2
Integrating Existing Applications with Oracle Identity Management	1-2
Integrating New Applications with Oracle Identity Management	1-3
Oracle Internet Directory Programming: An Overview	1-4
Programming Languages Supported by the Oracle Internet Directory SDK	1-4
Oracle Internet Directory SDK Components	1-4
Application Development in the Oracle Internet Directory Environment	1-4
Architecture of a Directory-Enabled Application	1-4
Oracle Internet Directory Interactions During the Application Life Cycle	1-5
Services and APIs for Integrating Applications with Oracle Internet Directory	1-6
Integrating Existing Applications with Oracle Internet Directory	1-8
Integrating New Applications with Oracle Internet Directory	1-8
Other Components of Oracle Internet Directory	1-9
2 Developing Applications with Standard LDAP APIs	
Sample Code	2-1
History of LDAP	2-1
LDAP Models	2-2
Naming Model	2-2
Information Model	2-3

Functional Model	2-3
Security Model.....	2-4
Authentication.....	2-4
Access Control and Authorization	2-5
Data Integrity.....	2-6
Data Privacy.....	2-6
Password Policies.....	2-6
About the Standard LDAP APIs.....	2-7
API Usage Model	2-7
Getting Started with the C API	2-7
Getting Started with the DBMS_LDAP Package.....	2-8
Getting Started with the Java API.....	2-8
Initializing an LDAP Session	2-8
Initializing the Session by Using the C API	2-8
Initializing the Session by Using DBMS_LDAP	2-9
Initializing the Session by Using JNDI.....	2-9
Authenticating an LDAP Session.....	2-10
Authenticating an LDAP Session by Using the C API	2-10
Authenticating an LDAP Session by Using DBMS_LDAP	2-11
Searching the Directory.....	2-11
Program Flow for Search Operations.....	2-12
Search Scope.....	2-13
Filters.....	2-14
Searching the Directory by Using the C API.....	2-15
Searching the Directory by Using DBMS_LDAP	2-16
Terminating the Session.....	2-17
Terminating the Session by Using the C API.....	2-17
Terminating the Session by Using DBMS_LDAP.....	2-17

3 Developing Applications with Oracle Extensions to the Standard APIs

Sample Code.....	3-1
Using Oracle Extensions to the Standard APIs	3-1
Creating an Application Identity in the Directory	3-2
Creating an Application Identity	3-2
Assigning Privileges to an Application Identity	3-3
Managing Users.....	3-3
Managing Groups	3-3
Managing Realms.....	3-3
Discovering a Directory Server.....	3-4
Benefits of Oracle Internet Directory Discovery Interfaces.....	3-4
Usage Model for Discovery Interfaces	3-5
Determining Server Name and Port Number From DNS.....	3-5
Mapping the DN of the Naming Context.....	3-6
Search by Domain Component of Local Machine.....	3-6
Search by Default SRV Record in DNS.....	3-6
Environment Variables for DNS Server Discovery.....	3-7
Programming Interfaces for DNS Server Discovery.....	3-7

SASL Authentication	3-7
SASL Authentication by Using the DIGEST-MD5 Mechanism.....	3-7
Steps Involved in SASL Authentication by Using DIGEST-MD5.....	3-8
SASL Authentication by Using External Mechanism.....	3-8
Proxying on Behalf of End Users	3-8
Creating Dynamic Password Verifiers	3-10
Request Control for Dynamic Password Verifiers.....	3-10
Syntax for DynamicVerifierRequestControl.....	3-10
Parameters Required by the Hashing Algorithms.....	3-11
Configuring the Authentication APIs.....	3-11
Parameters Passed If ldap_search Is Used.....	3-11
Parameters Passed If ldap_compare Is Used.....	3-12
Response Control for Dynamic Password Verifiers.....	3-12
Obtaining Privileges for the Dynamic Verifier Framework.....	3-12
4 Using the Java API Extensions to JNDI	
Sample Code	4-1
Installing the Java Extensions	4-1
Using the oracle.java.util Package to Model LDAP Objects	4-2
The Classes PropertySetCollection, PropertySet, and Property	4-2
Managing Users	4-3
Authenticating Users	4-3
Creating Users	4-4
Retrieving User Objects	4-4
Retrieving Objects from Realms	4-5
Discovering a Directory Server	4-5
Examples: Java API for Discovering a Directory Server	4-6
Using DIGEST-MD5 to Perform SASL Authentication	4-7
5 Using the API Extensions in PL/SQL	
Sample Code	5-1
Installing the PL/SQL Extensions	5-1
Using Handles to Access Directory Data	5-1
Managing Users	5-2
Authenticating Users	5-2
Dependencies and Limitations of the PL/SQL LDAP API	5-2
6 Developing Provisioning-Integrated Applications	
7 Developing Directory Plug-ins	
Plug-in Prerequisites	7-1
Plug-in Benefits	7-1
What Is the Plug-in Framework?	7-2
Operation-Based Plug-ins Supported by the Directory	7-2
Pre-Operation Plug-ins.....	7-2
Post-Operation Plug-ins.....	7-3

When-Operation Plug-ins	7-3
Designing, Creating, and Using Plug-ins	7-3
Designing Plug-ins	7-4
Types of Plug-in Operations	7-4
Naming Plug-ins	7-4
Creating Plug-ins	7-4
Package Specifications for Plug-in Module Interfaces	7-4
Compiling Plug-ins	7-6
Dependencies	7-6
Recompiling Plug-ins	7-6
Registering Plug-ins	7-6
The orclPluginConfig Object Class	7-6
Adding a Plug-in Configuration Entry by Using Command-Line Tools	7-8
Example 1	7-8
Example 2	7-9
Managing Plug-ins	7-9
Modifying Plug-ins	7-9
Debugging Plug-ins	7-9
Enabling and Disabling Plug-ins	7-9
Exception Handling	7-10
Error Handling	7-10
Program Control Handling between Oracle Internet Directory and Plug-ins	7-10
Plug-in LDAP API	7-11
Plug-ins and Replication	7-11
Plug-in and Database Tools	7-11
Security	7-12
Plug-in Debugging	7-12
Plug-in LDAP API Specifications	7-12
Database Limitations	7-13
Examples of Plug-ins	7-13
Example 1: Search Query Logging	7-13
Example 2: Synchronizing Two DITs	7-15
Binary Support in the Plug-in Framework	7-18
Binary Operations with ldapmodify	7-18
Binary Operations with ldapadd	7-20
Binary Operations with ldapcompare	7-22
Database Object Types Defined	7-25
Specifications for Plug-in Procedures	7-26

8 Integrating with Oracle Delegated Administration Services

What Is Oracle Delegated Administration Services?	8-1
How Applications Benefit from Oracle Delegated Administration Services	8-2
Integrating Applications with the Delegated Administration Services	8-2
Integration Profile	8-2
Integration Methodology and Considerations	8-2
Java APIs Used to Access URLs	8-4

9 Developing Applications for Single Sign-On

What Is mod_osso?	9-1
Protecting Applications Using mod_osso: Two Methods	9-2
Protecting URLs Statically	9-2
Protecting URLs with Dynamic Directives	9-2
Developing Applications Using mod_osso	9-3
Developing Statically Protected PL/SQL Applications	9-3
Developing Statically Protected Java Applications.....	9-5
Developing Java Applications That Use Dynamic Directives.....	9-6
Java Example #1: Simple Authentication	9-6
Java Example #2: Single Sign-Off	9-8
Java Example #3: Forced Authentication.....	9-8
A Word About Non-GET Authentication	9-9
Global Inactivity Timeout and Dynamic Directives	9-9
Security Issues	9-10
Single Sign-Off and Application Logout	9-10
Application Login: Code Examples.....	9-10
Application Logout: Recommended Code.....	9-12
Secure Transmission of mod_osso Cookies.....	9-12

10 Integrating J2EE Applications and Oracle Internet Directory

Standard J2EE Security APIs	10-1
OC4J Security APIs	10-2
JAAS Policy Management APIs	10-4
JAAS Policy Management.....	10-5
Retrieving User Policies and Permissions using Standard JAAS APIs.....	10-5

Part II Oracle Internet Directory Programming Reference

11 C API Reference

About the Oracle Internet Directory C API	11-1
Oracle Internet Directory SDK C API SSL Extensions.....	11-1
SSL Interface Calls	11-2
Wallet Support.....	11-2
Functions in the C API	11-2
The Functions at a Glance	11-3
Initializing an LDAP Session.....	11-5
ldap_init and ldap_open	11-5
LDAP Session Handle Options	11-6
ldap_get_option and ldap_set_option	11-6
Authenticating to the Directory	11-10
ldap_sasl_bind, ldap_sasl_bind_s, ldap_simple_bind, and ldap_simple_bind_s.....	11-10
SASL Authentication Using Oracle Extensions	11-12
ora_ldap_create_cred_hdl, ora_ldap_set_cred_props, ora_ldap_get_cred_props, and ora_ldap_free_cred_hdl.....	11-12
SASL Authentication	11-13

ora_ldap_init_SASL.....	11-14
Working With Controls.....	11-14
Closing the Session	11-16
ldap_unbind, ldap_unbind_ext, and ldap_unbind_s.....	11-16
Performing LDAP Operations.....	11-16
ldap_search_ext, ldap_search_ext_s, ldap_search, and ldap_search_s.....	11-17
Reading an Entry.....	11-19
Listing the Children of an Entry	11-20
ldap_compare_ext, ldap_compare_ext_s, ldap_compare, and ldap_compare_s	11-20
ldap_modify_ext, ldap_modify_ext_s, ldap_modify, and ldap_modify_s	11-21
ldap_rename and ldap_rename_s	11-23
ldap_add_ext, ldap_add_ext_s, ldap_add, and ldap_add_s	11-25
ldap_delete_ext, ldap_delete_ext_s, ldap_delete, and ldap_delete_s.....	11-26
ldap_extended_operation and ldap_extended_operation_s	11-28
Abandoning an Operation.....	11-29
ldap_abandon_ext and ldap_abandon	11-29
Obtaining Results and Peeking Inside LDAP Messages	11-30
ldap_result, ldap_msgtype, and ldap_msgid	11-30
Handling Errors and Parsing Results.....	11-32
ldap_parse_result, ldap_parse_sasl_bind_result, ldap_parse_extended_result, and ldap_err2string.....	11-32
Stepping Through a List of Results	11-34
ldap_first_message and ldap_next_message.....	11-34
Parsing Search Results.....	11-35
ldap_first_entry, ldap_next_entry, ldap_first_reference, ldap_next_reference, ldap_count_entries, and ldap_count_references	11-35
ldap_first_attribute and ldap_next_attribute.....	11-36
ldap_get_values, ldap_get_values_len, ldap_count_values, ldap_count_values_len, ldap_value_free, and ldap_value_free_len	11-37
ldap_get_dn, ldap_explode_dn, ldap_explode_rdn, and ldap_dn2ufn	11-38
ldap_get_entry_controls	11-39
ldap_parse_reference.....	11-39
Sample C API Usage	11-40
C API Usage with SSL	11-40
C API Usage Without SSL.....	11-41
C API Usage for SASL-Based DIGEST-MD5 Authentication.....	11-42
Required Header Files and Libraries for the C API	11-44
Dependencies and Limitations of the C API	11-45

12 DBMS_LDAP PL/SQL Reference

Summary of Subprograms	12-1
Exception Summary	12-3
Data Type Summary	12-5
Subprograms	12-5
FUNCTION init.....	12-5
FUNCTION simple_bind_s	12-6
FUNCTION bind_s.....	12-7

FUNCTION unbind_s	12-8
FUNCTION compare_s.....	12-9
FUNCTION search_s.....	12-10
FUNCTION search_st.....	12-12
FUNCTION first_entry	12-13
FUNCTION next_entry	12-14
FUNCTION count_entries.....	12-15
FUNCTION first_attribute.....	12-16
FUNCTION next_attribute	12-17
FUNCTION get_dn.....	12-18
FUNCTION get_values.....	12-19
FUNCTION get_values_len.....	12-20
FUNCTION delete_s.....	12-21
FUNCTION modrdn2_s.....	12-22
FUNCTION err2string.....	12-23
FUNCTION create_mod_array	12-24
PROCEDURE populate_mod_array (String Version)	12-25
PROCEDURE populate_mod_array (Binary Version)	12-25
PROCEDURE populate_mod_array (Binary Version. Uses BLOB Data Type)	12-26
FUNCTION get_values_blob	12-27
FUNCTION count_values_blob.....	12-28
FUNCTION value_free_blob.....	12-29
FUNCTION modify_s	12-29
FUNCTION add_s	12-30
PROCEDURE free_mod_array.....	12-31
FUNCTION count_values	12-32
FUNCTION count_values_len.....	12-32
FUNCTION rename_s.....	12-33
FUNCTION explode_dn.....	12-34
FUNCTION open_ssl.....	12-35
FUNCTION msgfree.....	12-36
FUNCTION ber_free	12-37
FUNCTION nls_convert_to_utf8.....	12-38
FUNCTION nls_convert_to_utf8.....	12-38
FUNCTION nls_convert_from_utf8.....	12-39
FUNCTION nls_convert_from_utf8.....	12-40
FUNCTION nls_get_dbcharset_name	12-41

13 Java API Reference

14 DBMS_LDAP_UTL PL/SQL Reference

Summary of Subprograms	14-1
Subprograms	14-2
User-Related Subprograms.....	14-3
Function authenticate_user	14-3
Function create_user_handle	14-5

Function set_user_handle_properties.....	14-5
Function get_user_properties.....	14-6
Function set_user_properties.....	14-7
Function get_user_extended_properties.....	14-9
Function get_user_dn.....	14-10
Function check_group_membership.....	14-11
Function locate_subscriber_for_user.....	14-12
Function get_group_membership.....	14-13
Group-Related Subprograms.....	14-13
Function create_group_handle.....	14-14
Function set_group_handle_properties.....	14-15
Function get_group_properties.....	14-16
Function get_group_dn.....	14-17
Subscriber-Related Subprograms.....	14-18
Function create_subscriber_handle.....	14-19
Function get_subscriber_properties.....	14-19
Function get_subscriber_dn.....	14-21
Function get_subscriber_ext_properties.....	14-22
Property-Related Subprograms.....	14-23
Miscellaneous Subprograms.....	14-24
Function normalize_dn_with_case.....	14-24
Function get_property_names.....	14-24
Function get_property_values.....	14-25
Function get_property_values_len.....	14-26
Procedure free_propertyset_collection.....	14-27
Function create_mod_propertyset.....	14-28
Function populate_mod_propertyset.....	14-29
Procedure free_mod_propertyset.....	14-29
Procedure free_handle.....	14-30
Function check_interface_version.....	14-30
Function get_property_values_blob.....	14-31
Procedure property_value_free_blob.....	14-32
Function Return Code Summary.....	14-32
Data Type Summary.....	14-34

15 DAS_URL Interface Reference

Directory Entries for the Service Units.....	15-1
Service Units and Corresponding URL Parameters.....	15-2
DAS URL API Parameter Descriptions.....	15-5
Search-and-Select Service Units for Users or Groups.....	15-6
Invoking Search-and-Select Service Units for Users or Groups.....	15-6
Receiving Data from the User or Group Search-and-Select Service Units.....	15-7

16 Centralized User Provisioning Java API Reference

Application Configuration.....	16-1
Application Registration and Provisioning Configuration.....	16-1
Application Registration.....	16-2

Provisioning Configuration.....	16-4
Application Configuration Classes.....	16-12
User Management	16-13
Creating a User	16-13
Modifying a User.....	16-14
Deleting a User	16-14
Looking Up a User	16-14
Debugging	16-14
Sample Code	16-15

17 Provisioning Integration PL/SQL API Reference

Versioning of Provisioning Files and Interfaces	17-1
Extensible Event Definition Configuration	17-1
Inbound and Outbound Events	17-3
PL/SQL Bidirectional Interface (Version 3.0)	17-4
PL/SQL Bidirectional Interface (Version 2.0)	17-8
Provisioning Event Interface (Version 1.1)	17-9
Predefined Event Types	17-11
Attribute Type	17-11
Attribute Modification Type.....	17-11
Event Dispositions Constants.....	17-11
Callbacks.....	17-11
GetAppEvent()	17-12
PutAppEventStatus().....	17-12
PutOIDEvent()	17-12

Part III Appendixes

A Java Plug-ins for User Provisioning

Plug-in Types and Their Purpose	A-1
Plug-in Requirements	A-2
Data Entry Plug-in	A-2
Pre-Data-Entry Plug-in	A-4
Post-Data-Entry Plug-in	A-5
Data Access Plug-in	A-5
Event Delivery Plug-in	A-7
Plug-in Return Status	A-10
Configuration Template	A-10
Sample Code	A-11

B DSML Syntax

Capabilities of DSML	B-1
Benefits of DSML	B-1
DSML Syntax	B-1
Top-Level Structure	B-2
Directory Entries	B-2

Schema Entries.....	B-3
Tools Enabled for DSML	B-3

Glossary

Index

List of Figures

1-1	A Directory-Enabled Application.....	1-5
1-2	An Application Leveraging APIs and Services	1-7
2-1	A Directory Information Tree	2-2
2-2	Attributes of the Entry for Anne Smith	2-3
2-3	Steps in Typical DBMS_LDAP Usage	2-7
2-4	Flow of Search-Related Operations.....	2-13
2-5	The Three Scope Options.....	2-14
3-1	Programmatic Flow for API Extensions	3-2
8-1	Overview of Delegated Administration Services.....	8-1
16-1	The Directory Information Tree for Provisioning Configuration Data	16-5

List of Tables

1-1	Interactions During Application Lifecycle	1-5
1-2	Services and APIs for Integrating with Oracle Internet Directory	1-6
1-3	Services for Modifying Existing Applications	1-8
1-4	Application Integration Points.....	1-9
2-1	LDAP Functions	2-4
2-2	SSL Authentication Modes	2-5
2-3	Parameters for ldap_init()	2-9
2-4	Arguments for ldap_simple_bind_s()	2-11
2-5	Options for search_s() or search_st() Functions	2-13
2-6	Search Filters.....	2-14
2-7	Boolean Operators	2-15
2-8	Arguments for ldap_search_s().....	2-16
2-9	Arguments for DBMS_LDAP.search_s() and DBMS_LDAP.search_st()	2-16
3-1	Environment Variables for DNS Discovery	3-7
3-2	Parameters in DynamicVerifierRequestControl.....	3-11
3-3	Parameters Required by the Hashing Algorithms.....	3-11
4-1	Methods for Directory Server Discovery.....	4-5
7-1	Plug-in Module Interface	7-4
7-2	Operation-Based and Attribute-Based Plug-in Procedure Signatures.....	7-5
7-3	Plug-in Attribute Names and Values.....	7-7
7-4	Valid Values for the plug-in Return Code	7-10
7-5	Program Control Handling when a Plug-in Exception Occurs	7-10
7-6	Program Control Handling when an LDAP Operation Fails.....	7-11
8-1	Integration Considerations	8-2
8-2	URL Parameters for Oracle Delegated Administration Services.....	8-3
9-1	User Attributes Passed to Partner Applications.....	9-1
9-2	Commonly Requested Dynamic Directives.....	9-3
11-1	Arguments for SSL Interface Calls	11-2
11-2	Functions and Procedures in the C API.....	11-3
11-3	Parameters for Initializing an LDAP Session.....	11-5
11-4	Parameters for LDAP Session Handle Options.....	11-7
11-5	Constants.....	11-7
11-6	Parameters for Authenticating to the Directory	11-11
11-7	Parameters for Managing SASL Credentials	11-13
11-8	Parameters for Managing SASL Credentials	11-14
11-9	Fields in ldapcontrol Structure	11-14
11-10	Parameters for Closing the Session	11-16
11-11	Parameters for Search Operations	11-18
11-12	Parameters for Compare Operations	11-21
11-13	Parameters for Modify Operations.....	11-22
11-14	Fields in LDAPMod Structure.....	11-23
11-15	Parameters for Rename Operations	11-25
11-16	Parameters for Add Operations.....	11-26
11-17	Parameters for Delete Operations	11-27
11-18	Parameters for Extended Operations.....	11-29
11-19	Parameters for Abandoning an Operation.....	11-30
11-20	Parameters for Obtaining Results and Peeking Inside LDAP Messages.....	11-31
11-21	Parameters for Handling Errors and Parsing Results	11-33
11-22	Parameters for Stepping Through a List of Results	11-34
11-23	Parameters for Retrieving Entries and Continuation References from a Search Result Chain, and for Counting Entries Returned	11-35
11-24	Parameters for Stepping Through Attribute Types Returned with an Entry	11-36
11-25	Parameters for Retrieving and Counting Attribute Values.....	11-37

11-26	Parameters for Retrieving, Exploding, and Converting Entry Names	11-38
11-27	Parameters for Extracting LDAP Controls from an Entry	11-39
11-28	Parameters for Extracting Referrals and Controls from a SearchResultReference Message ..	11-40
12-1	DBMS_LDAP API Subprograms	12-1
12-2	DBMS_LDAP Exception Summary	12-3
12-3	DBMS_LDAP Data Type Summary	12-5
12-4	INIT Function Parameters	12-5
12-5	INIT Function Return Values	12-6
12-6	INIT Function Exceptions	12-6
12-7	SIMPLE_BIND_S Function Parameters	12-7
12-8	SIMPLE_BIND_S Function Return Values.....	12-7
12-9	SIMPLE_BIND_S Function Exceptions.....	12-7
12-10	BIND_S Function Parameters	12-7
12-11	BIND_S Function Return Values	12-8
12-12	BIND_S Function Exceptions	12-8
12-13	UNBIND_S Function Parameters	12-8
12-14	UNBIND_S Function Return Values.....	12-9
12-15	UNBIND_S Function Exceptions.....	12-9
12-16	COMPARE_S Function Parameters	12-9
12-17	COMPARE_S Function Return Values.....	12-10
12-18	COMPARE_S Function Exceptions	12-10
12-19	SEARCH_S Function Parameters	12-10
12-20	SEARCH_S Function Return Value.....	12-11
12-21	SEARCH_S Function Exceptions.....	12-11
12-22	SEARCH_ST Function Parameters.....	12-12
12-23	SEARCH_ST Function Return Values	12-13
12-24	SEARCH_ST Function Exceptions	12-13
12-25	FIRST_ENTRY Function Parameters	12-13
12-26	FIRST_ENTRY Return Values.....	12-14
12-27	FIRST_ENTRY Exceptions.....	12-14
12-28	NEXT_ENTRY Function Parameters	12-14
12-29	NEXT_ENTRY Function Return Values.....	12-15
12-30	NEXT_ENTRY Function Exceptions.....	12-15
12-31	COUNT_ENTRY Function Parameters	12-15
12-32	COUNT_ENTRY Function Return Values	12-16
12-33	COUNT_ENTRY Function Exceptions.....	12-16
12-34	FIRST_ATTRIBUTE Function Parameters.....	12-16
12-35	FIRST_ATTRIBUTE Function Return Values	12-17
12-36	FIRST_ATTRIBUTE Function Exceptions	12-17
12-37	NEXT_ATTRIBUTE Function Parameters	12-17
12-38	NEXT_ATTRIBUTE Function Return Values	12-18
12-39	NEXT_ATTRIBUTE Function Exceptions	12-18
12-40	GET_DN Function Parameters	12-18
12-41	GET_DN Function Return Values	12-19
12-42	GET_DN Function Exceptions	12-19
12-43	GET_VALUES Function Parameters.....	12-19
12-44	GET_VALUES Function Return Values	12-20
12-45	GET_VALUES Function Exceptions	12-20
12-46	GET_VALUES_LEN Function Parameters.....	12-20
12-47	GET_VALUES_LEN Function Return Values	12-21
12-48	GET_VALUES_LEN Function Exceptions	12-21
12-49	DELETE_S Function Parameters	12-21
12-50	DELETE_S Function Return Values.....	12-22
12-51	DELETE_S Function Exceptions	12-22

12-52	MODRDN2_S Function Parameters.....	12-22
12-53	MODRDN2_S Function Return Values	12-23
12-54	MODRDN2_S Function Exceptions	12-23
12-55	ERR2STRING Function Parameters	12-23
12-56	ERR2STRING Function Return Values.....	12-24
12-57	CREATE_MOD_ARRAY Function Parameters.....	12-24
12-58	CREATE_MOD_ARRAY Function Return Values	12-24
12-59	POPULATE_MOD_ARRAY (String Version) Procedure Parameters	12-25
12-60	POPULATE_MOD_ARRAY (String Version) Procedure Exceptions	12-25
12-61	POPULATE_MOD_ARRAY (Binary Version) Procedure Parameters	12-26
12-62	POPULATE_MOD_ARRAY (Binary Version) Procedure Exceptions	12-26
12-63	POPULATE_MOD_ARRAY (Binary) Parameters	12-27
12-64	POPULATE_MOD_ARRAY (Binary) Exceptions	12-27
12-65	GET_VALUES_BLOB Parameters	12-27
12-66	get_values_blob Return Values.....	12-28
12-67	get_values_blob Exceptions.....	12-28
12-68	COUNT_VALUES_BLOB Parameters	12-28
12-69	COUNT_VALUES_BLOB Return Values.....	12-29
12-70	VALUE_FREE_BLOB Parameters	12-29
12-71	MODIFY_S Function Parameters	12-30
12-72	MODIFY_S Function Return Values	12-30
12-73	MODIFY_S Function Exceptions	12-30
12-74	ADD_S Function Parameters	12-31
12-75	ADD_S Function Return Values	12-31
12-76	ADD_S Function Exceptions.....	12-31
12-77	FREE_MOD_ARRAY Procedure Parameters	12-32
12-78	COUNT_VALUES Function Parameters.....	12-32
12-79	COUNT_VALUES Function Return Values	12-32
12-80	COUNT_VALUES_LEN Function Parameters.....	12-33
12-81	COUNT_VALUES_LEN Function Return Values	12-33
12-82	RENAME_S Function Parameters	12-33
12-83	RENAME_S Function Return Values.....	12-34
12-84	RENAME_S Function Exceptions.....	12-34
12-85	EXPLODE_DN Function Parameters.....	12-34
12-86	EXPLODE_DN Function Return Values	12-35
12-87	EXPLODE_DN Function Exceptions	12-35
12-88	OPEN_SSL Function Parameters.....	12-35
12-89	OPEN_SSL Function Return Values	12-36
12-90	OPEN_SSL Function Exceptions.....	12-36
12-91	MSGFREE Function Parameters	12-36
12-92	MSGFREE Return Values	12-37
12-93	BER_FREE Function Parameters	12-37
12-94	Parameters for nls_convert_to_utf8	12-38
12-95	Return Values for nls_convert_to_utf8.....	12-38
12-96	Parameters for nls_convert_to_utf8	12-39
12-97	Return Values for nls_convert_to_utf8.....	12-39
12-98	Parameter for nls_convert_from_utf8.....	12-39
12-99	Return Value for nls_convert_from_utf8.....	12-39
12-100	Parameter for nls_convert_from_utf8.....	12-40
12-101	Return Value for nls_convert_from_utf8.....	12-40
12-102	Return Value for nls_get_dbcharset_name	12-41
14-1	DBMS_LDAP_UTL User-Related Subprograms	14-1
14-2	DBMS_LDAP_UTL Group-Related Subprograms.....	14-2
14-3	DBMS_LDAP_UTL Subscriber-Related Subprograms.....	14-2
14-4	DBMS_LDAP_UTL Miscellaneous Subprograms.....	14-2

14-5	authenticate_user Function Parameters	14-4
14-6	authenticate_user Function Return Values	14-4
14-7	CREATE_USER_HANDLE Function Parameters.....	14-5
14-8	CREATE_USER_HANDLE Function Return Values	14-5
14-9	SET_USER_HANDLE_PROPERTIES Function Parameters.....	14-6
14-10	SET_USER_HANDLE_PROPERTIES Function Return Values	14-6
14-11	GET_USER_PROPERTIES Function Parameters	14-6
14-12	GET_USER_PROPERTIES Function Return Values	14-7
14-13	SET_USER_PROPERTIES Function Parameters	14-8
14-14	SET_USER_PROPERTIES Function Return Values	14-8
14-15	GET_USER_EXTENDED_PROPERTIES Function Parameters	14-9
14-16	GET_USER_EXTENDED_PROPERTIES Function Return Values	14-9
14-17	GET_USER_DN Function Parameters	14-10
14-18	GET_USER_DN Function Return Values.....	14-10
14-19	CHECK_GROUP_MEMBERSHIP Function Parameters	14-11
14-20	CHECK_GROUP_MEMBERSHIP Function Return Values	14-11
14-21	LOCATE_SUBSCRIBER_FOR_USER Function Parameters.....	14-12
14-22	LOCATE SUBSCRIBER FOR USER Function Return Values	14-12
14-23	GET_GROUP_MEMBERSHIP Function Parameters.....	14-13
14-24	GET_GROUP_MEMBERSHIP Function Return Values	14-13
14-25	CREATE_GROUP_HANDLE Function Parameters.....	14-15
14-26	CREATE_GROUP_HANDLE Function Return Values	14-15
14-27	SET_GROUP_HANDLE_PROPERTIES Function Parameters.....	14-15
14-28	SET_GROUP_HANDLE_PROPERTIES Function Return Values	14-16
14-29	GET_GROUP_PROPERTIES Function Parameters	14-16
14-30	GET_GROUP_PROPERTIES Function Return Values.....	14-17
14-31	GET_GROUP_DN Function Parameters	14-18
14-32	GET_GROUP_DN Function Return Values.....	14-18
14-33	CREATE_SUBSCRIBER_HANDLE Function Parameters.....	14-19
14-34	CREATE_SUBSCRIBER_HANDLE Function Return Values.....	14-19
14-35	GET_SUBSCRIBER_PROPERTIES Function Parameters.....	14-20
14-36	GET_SUBSCRIBER_PROPERTIES Function Return Values	14-20
14-37	GET_SUBSCRIBER_DN Function Parameters	14-21
14-38	GET_SUBSCRIBER_DN Function Return Values.....	14-21
14-39	GET_SUBSCRIBER_EXT_PROPERTIES Function Parameters	14-22
14-40	GET_USER_EXTENDED_PROPERTIES Function Return Values	14-22
14-41	NORMALIZE_DN_WITH_CASE Function Parameters.....	14-24
14-42	NORMALIZE_DN_WITH_CASE Function Return Values	14-24
14-43	GET_PROPERTY_NAMES Function Parameters	14-25
14-44	GET_PROPERTY_NAMES Function Return Values.....	14-25
14-45	GET_PROPERTY_VALUES Function Parameters.....	14-25
14-46	GET_PROPERTY_VALUES Function Return Values.....	14-26
14-47	GET_PROPERTY_VALUES_LEN Function Parameters.....	14-26
14-48	GET_PROPERTY_VALUES_LEN Function Return Values	14-27
14-49	FREE_PROPERTYSET_COLLECTION Procedure Parameters	14-28
14-50	CREATE_MOD_PROPERTYSET Function Parameters	14-28
14-51	CREATE_MOD_PROPERTYSET Function Return Values.....	14-28
14-52	POPULATE_MOD_PROPERTYSET Function Parameters	14-29
14-53	POPULATE_MOD_PROPERTYSET Function Return Values	14-29
14-54	FREE_MOD_PROPERTYSET Procedure Parameters.....	14-30
14-55	FREE_HANDLE Procedure Parameters.....	14-30
14-56	CHECK_INTERFACE_VERSION Function Parameters.....	14-30
14-57	CHECK_VERSION_INTERFACE Function Return Values	14-31
14-58	GET_PROPERTY_VALUES_BLOB Function Parameters	14-31
14-59	GET_PROPERTY_VALUES_BLOB Return Values.....	14-31

14-60	PROPERTY_VALUE_FREE_BLOB Function Parameters.....	14-32
14-61	Function Return Codes	14-32
14-62	DBMS_LDAP_UTL Data Types.....	14-34
15-1	Service Units and Corresponding Entries	15-1
15-2	Service Units and Corresponding URL Parameters	15-2
15-3	DAS URL Parameter Descriptions	15-5
15-4	User Search and Select.....	15-7
15-5	Group Search and Select	15-7
16-1	Some Useful Privilege Groups	16-3
16-2	Interfaces and Their Configuration	16-7
16-3	Information Formats Supported by the PLSQL Interface.....	16-8
16-4	Properties Stored as Attributes in the Attribute Configuration Entry.....	16-9
16-5	Event propagation parameters.....	16-11
17-1	Predefined Event Definitions	17-2
17-2	Attributes of the Provisioning Subscription Profile.....	17-4

Preface

Oracle Identity Management Application Developer's Guide explains how to modify applications to work with the Oracle Identity Management infrastructure. For the purposes of this book, this infrastructure consists of Oracle Application Server Single Sign-On, Oracle Internet Directory, Oracle Delegated Administration Services, and the Directory Integration Platform.

This preface contains these topics:

- [Audience](#)
- [Documentation Accessibility](#)
- [Related Documents](#)
- [Conventions](#)

Audience

The following readers can benefit from this book:

- Developers who want to integrate applications with the Oracle Identity Management infrastructure. This process involves storing and updating information in an Oracle Internet Directory server. It also involves modifying applications to work with `mod_osso`, an authentication module on the Oracle HTTP Server.
- Anyone who wants to learn about the LDAP APIs and Oracle extensions to these APIs.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

<http://www.oracle.com/accessibility/>

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documents

For more information, see these Oracle resources:

- *Oracle Identity Management Concepts and Deployment Planning Guide*
- *Oracle Internet Directory Administrator's Guide*
- *Oracle Identity Management Integration Guide*
- *Oracle Identity Management Guide to Delegated Administration*
- *Oracle Application Server Single Sign-On Administrator's Guide*
- *PL/SQL User's Guide and Reference*
- *Oracle Database Application Developer's Guide - Fundamentals*

For additional information, see:

- Chadwick, David. *Understanding X.500—The Directory*. Thomson Computer Press, 1996.
- Howes, Tim and Mark Smith. *LDAP: Programming Directory-enabled Applications with Lightweight Directory Access Protocol*. Macmillan Technical Publishing, 1997.
- Howes, Tim, Mark Smith and Gordon Good, *Understanding and Deploying LDAP Directory Services*. Macmillan Technical Publishing, 1999.
- Internet Assigned Numbers Authority home page, <http://www.iana.org>, for information about object identifiers
- Internet Engineering Task Force (IETF) documentation available at: <http://www.ietf.org>, especially:
 - The LDAPEXT charter and LDAP drafts
 - The LDUP charter and drafts
 - RFC 2254, "The String Representation of LDAP Search Filters"
 - RFC 1823, "The LDAP Application Program Interface"
- The OpenLDAP Community, <http://www.openldap.org>

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
<code>monospace</code>	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in the SDK?

This document acquaints you with new features in the Oracle Internet Directory Software Developer's Kit—both in the present release and in the last release. Use the links provided to learn more about each feature.

New Features in the Release 10.1.2 SDK

The release 10.1.2 SDK adds:

- Centralized user provisioning.
This feature enables you to provision application users into the Oracle Identity Management infrastructure. To learn more, see [Chapter 16, "Centralized User Provisioning Java API Reference"](#).
- Dynamic password verifiers
This feature addresses the needs of applications that provide parameters for password verifiers only at runtime. To learn more, see "[Creating Dynamic Password Verifiers](#)" in [Chapter 3](#).
- Binary support for `ldapmodify`, `ldapadd`, and `ldapcompare` plug-ins
Directory plug-ins can now access binary attributes in the directory database. To learn more, see "[Binary Support in the Plug-in Framework](#)" in [Chapter 7](#).
- Plug-in support for the Oracle Directory Integration and Provisioning Server
These Java hooks enable an enterprise to incorporate its own business rules and to tailor footprint creation to its needs. To learn more, see [Appendix A](#).

New Features in the Release 9.0.4 SDK

The following features made their debut in the release 9.0.4 SDK:

- URL API for Oracle Delegated Administration Services
This API enables you to build administrative and self-service consoles that delegated administrators can use to perform directory operations. To learn more, see [Chapter 8](#).
- PL/SQL API Enhancements:
 - New functions in the LDAP v3 standard. Previously available only in the C API, these functions are now available in PL/SQL.
 - Functions that enable proxied access to middle-tier applications.

- Functions that create and manage provisioning profiles in the directory integration and provisioning platform.

To learn more, see [Chapter 6](#).

- Plug-in support for external authentication

This feature enables administrators to use Microsoft Active Directory to store and manage security credentials for Oracle components. To learn more, see [Chapter 7](#).

- Server discovery using DNS

This feature enables directory clients to discover the host name and port number of a directory server. It reduces the cost of maintaining directory clients in large deployments. To learn more, see "[Discovering a Directory Server](#)" in [Chapter](#) .

- XML support for the directory SDK and directory tools

This feature enables LDAP tools to process XML as well as LDIF notation. Directory APIs can manipulate data in a DSML 1.0 format.

- Caching for client-side referrals

This feature enables clients to cache referral information, speeding up referral processing. To learn more, see "[LDAP Session Handle Options](#)" in [Chapter 8](#).

Part I

Programming for Oracle Identity Management

Part I shows you how to modify your applications to work with the different components of Oracle Identity Management. This section begins with an introduction to the Oracle Internet Directory SDK and to LDAP programming concepts. You then learn how to use the three LDAP APIs and their extensions to enable applications for Oracle Internet Directory. The section ends with the tasks required to enable an application for single sign-on.

Part I contains these chapters:

- [Chapter 1, "Developing Applications for Oracle Identity Management"](#)
- [Chapter 2, "Developing Applications with Standard LDAP APIs"](#)
- [Chapter 3, "Developing Applications with Oracle Extensions to the Standard APIs"](#)
- [Chapter 6, "Developing Provisioning-Integrated Applications"](#)
- [Chapter 7, "Developing Directory Plug-ins"](#)
- [Chapter 8, "Integrating with Oracle Delegated Administration Services"](#)
- [Chapter 9, "Developing Applications for Single Sign-On"](#)
- [Chapter 10, "Integrating J2EE Applications and Oracle Internet Directory"](#)

Developing Applications for Oracle Identity Management

Oracle Identity Management provides a shared infrastructure for all Oracle applications. It also provides services and interfaces that facilitate third-party enterprise application development. These interfaces are useful for application developers who need to incorporate identity management into their applications.

This chapter discusses these interfaces and recommends application development best practices in the Oracle Identity Management environment.

There are two types of applications that can be integrated with Oracle Identity Management:

- Existing applications already used in the enterprise. The enterprise might have already invested in such applications and would benefit from their integration with the Oracle Identity Management infrastructure.
- New applications being developed by corporate IT departments or ISVs that are based on the Oracle technology stack

This chapter contains the following topics:

- [Benefits of Integrating with Oracle Identity Management](#)
- [Oracle Identity Management Services Available for Application Integration](#)
- [Integrating Existing Applications with Oracle Identity Management](#)
- [Integrating New Applications with Oracle Identity Management](#)
- [Oracle Internet Directory Programming: An Overview](#)

Benefits of Integrating with Oracle Identity Management

Enterprise applications integrating with the Oracle Identity Management infrastructure receive the following benefits:

- **Integration facilitates faster application deployment with lower costs:** Enterprises (primarily Oracle customers) already using an existing Oracle Identity Management infrastructure can deploy new applications using the self-service console of Oracle Delegated Administration Services. Delegating application administration to users reduces the deployment cost of the application.
- **Seamless integration with Oracle applications:** Because all Oracle applications rely on the Oracle Identity Management infrastructure, new enterprise applications can use all the features Oracle Identity Management offers.

- **Seamless integration with third-party identity management solutions:** Because the Oracle Identity Management infrastructure already has built-in capabilities for integrating with third-party identity management solutions, application developers can take advantage of the identity management features.

Oracle Identity Management Services Available for Application Integration

Custom applications can use Oracle Identity Management through a set of documented and supported services and APIs. For example:

- Oracle Internet Directory provides LDAP APIs for C, Java, and PL/SQL, and is compatible with other LDAP SDKs.
- Oracle Delegated Administration Services provides a core self-service console that can be customized to support third-party applications. In addition, they provide a number of services for building customized administration interfaces that manipulate directory data.
- Oracle Directory Integration Services facilitate the development and deployment of custom solutions for synchronizing Oracle Internet Directory with third-party directories and other user repositories.
- Oracle Provisioning Integration Services provide a mechanism for provisioning third-party applications, as well as a means of integrating the Oracle environment with other provisioning systems.
- OracleAS Single Sign-On provides APIs for developing and deploying partner applications that share a single sign-on session with other Oracle Web applications.
- JAZN is the Oracle implementation of the Java Authentication and Authorization Service (JAAS) Support standard. JAZN allows applications developed for the Web using the Oracle J2EE environment to use the identity management infrastructure for authentication and authorization.

Integrating Existing Applications with Oracle Identity Management

An enterprise may have already deployed certain applications to perform critical business functions. The Oracle Identity Management infrastructure provides the following services that can be leveraged by the deployment to modify existing applications:

- **Automated User Provisioning:** The deployment can develop a custom provisioning agent that automates the provisioning of users in the existing application in response to provisioning events in the Oracle Identity Management infrastructure. This agent must be developed using the interfaces of Oracle Provisioning Integration Service.

See Also: *Oracle Internet Directory Administrator's Guide* for more information about developing automated user provisioning

- **User Authentication Services:** If the user interface of the existing application is based on HTTP, integrating it with Oracle HTTP Server and protecting its URL using `mod_ossso` will authenticate all incoming user requests using the OracleAS Single Sign-On service.
- **Centralized User Profile Management:** If the user interface of the existing application is based on HTTP, and it is integrated with OracleAS Single Sign-On for authentication, the application can use the self-service console of Oracle

Delegated Administration Services to enable centralized user profile management. The self-service console can be customized by the deployment to address the specific needs of the application.

Integrating New Applications with Oracle Identity Management

Application developers can use the services provided by the Oracle Identity Management infrastructure more extensively if they are developing a new application or planning a new release of an existing application. Application developers should consider the following integration points:

- **User Authentication Services:** The application developer has the following options:
 - If the application is based on J2EE, it can use the services provided by the Oracle Application Server Java Authentication and Authorization Service (JAAS) Provider interface.
 - If the application relies on Oracle Application Server Containers for J2EE (OC4J), it can use the services provided by `mod_ossso` to authenticate users and obtain important information about the user in the HTTP headers.
 - If the application is a standalone Web-based application, it can use OracleAS Single Sign-On as a partner application using the OracleAS Single Sign-On APIs.
 - If the application provides an interface that is not Web-based, it can use the Oracle Internet Directory LDAP APIs (available in C, PL/SQL and Java) to authenticate users.
- **Centralized Profile Management:** The application developer has the following options available:
 - The application developer can model application-specific profiles and user preferences as attributes in Oracle Internet Directory.
 - If the user interface of the application is based on HTTP, and it is integrated with OracleAS Single Sign-On for authentication, the application can leverage the self-service console of Oracle Delegated Administration Services to enable centralized user profile management. The self-service console can be customized by the deployment to address the specific needs of the application.
 - The application can also retrieve user profiles at run time using the Oracle Internet Directory LDAP APIs (available in C, PL/SQL and Java).
- **Automated User Provisioning:** Application developers should consider the following options:
 - If the user interface of the application is based on HTTP and it is integrated with OracleAS Single Sign-On for authentication, then the application developer can implement automated user provisioning the first time a user accesses the application
 - The application can also be integrated with the Oracle Internet Directory Provisioning Integration Service, which enables it to automatically provision or de-provision user accounts in response to administrative actions, such as adding an identity, modifying the properties of an existing identity, or deleting an existing identity in the Oracle Identity Management infrastructure

See Also: *Oracle Identity Management Integration Guide*

Oracle Internet Directory Programming: An Overview

This section introduces you to the Oracle Internet Directory Software Developer's Kit. It provides an overview of how an application can use the kit to integrate with the directory. You are also acquainted with the rest of the directory product suite.

The section contains these topics:

- [Programming Languages Supported by the Oracle Internet Directory SDK](#)
- [Oracle Internet Directory SDK Components](#)
- [Application Development in the Oracle Internet Directory Environment](#)
- [Other Components of Oracle Internet Directory](#)

Programming Languages Supported by the Oracle Internet Directory SDK

The SDK is for application developers who use C, C++, and PL/SQL. Java developers must use the JNDI provider from Sun Microsystems to integrate with the directory.

Oracle Internet Directory SDK Components

Oracle Internet Directory Software Developer's Kit 10g Release 2 (10.1.2) consists of the following:

- A C API compliant with LDAP Version 3
- A PL/SQL API contained in a PL/SQL package called `DBMS_LDAP`
- Sample programs
- *Oracle Identity Management Application Developer's Guide* (this document)
- Command-line tools

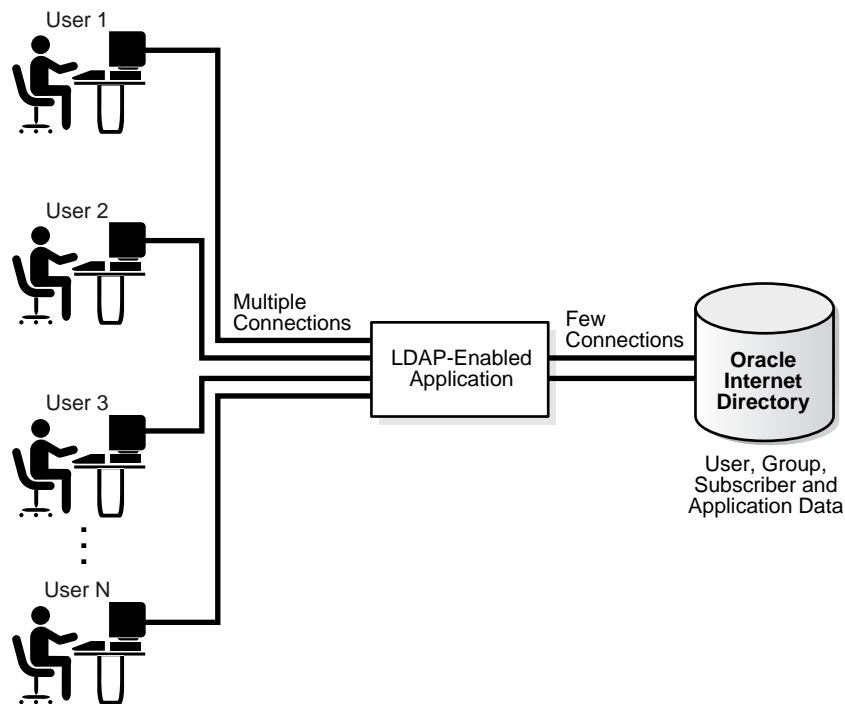
Application Development in the Oracle Internet Directory Environment

This section contains these topics:

- [Architecture of a Directory-Enabled Application](#)
- [Oracle Internet Directory Interactions During the Application Life Cycle](#)
- [Services and APIs for Integrating Applications with Oracle Internet Directory](#)
- [Integrating Existing Applications with Oracle Internet Directory](#)
- [Integrating New Applications with Oracle Internet Directory](#)

Architecture of a Directory-Enabled Application

Most directory-enabled applications are backend programs that simultaneously handle multiple requests from multiple users. [Figure 1-1](#) shows how a directory is used by such applications.

Figure 1-1 A Directory-Enabled Application

As [Figure 1-1](#) shows, when a user request involves an LDAP-enabled operation, the application processes the request using a smaller set of pre-created directory connections.

Oracle Internet Directory Interactions During the Application Life Cycle

[Table 1-1](#) on page 1-5 walks you through the directory operations that an application typically performs during its lifecycle.

Table 1-1 Interactions During Application Lifecycle

Point in Application Lifecycle	Logic
Application Installation	<ol style="list-style-type: none"> 1. Create an application identity in the directory. The application uses this identity to perform most of its LDAP operations. 2. Give the application identity LDAP authorizations by making it part of the correct LDAP groups. These authorizations enable the application to accept user credentials and authenticate them against the directory. The directory can also use application authorizations to proxy for the user when LDAP operations must be performed on the user's behalf.
Application Startup and Bootstrap	<p>The application must retrieve credentials that enable it to authenticate itself to the directory.</p> <p>If the application stores configuration metadata in Oracle Internet Directory, it can retrieve that metadata and initialize other parts of the application.</p> <p>The application can then establish a pool of connections to serve user requests.</p>

Table 1–1 (Cont.) Interactions During Application Lifecycle

Point in Application Lifecycle	Logic
Application Runtime	<p>For every end-user request that needs an LDAP operation, the application can:</p> <ul style="list-style-type: none"> ▪ Pick a connection from the pool of LDAP connections. ▪ Switch the user to the end-user identity if the LDAP operation needs to be performed with the effective rights of the end-user. ▪ Perform the LDAP operation by using either the regular API or the API enhancements described in this chapter. ▪ Ensure that the effective user is now the application identity once the LDAP operation is complete. ▪ Return the LDAP connection back to the pool of connections.
Application Shutdown	Abandon any outstanding LDAP operations and close all LDAP connections.
Application Deinstallation	Remove the application identity and the LDAP authorizations granted to it.

Services and APIs for Integrating Applications with Oracle Internet Directory

Application developers can integrate with Oracle Internet Directory by using the services and APIs listed and described in [Table 1–2](#) on page 1-6.

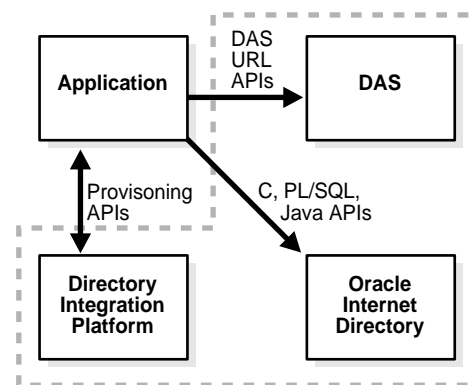
Table 1–2 Services and APIs for Integrating with Oracle Internet Directory

Service/API	Description	More Information
Standard LDAP APIs in C, PL/SQL and Java	These provide basic LDAP operations. The standard LDAP API used in Java is the JNDI API with the LDAP service provider from Sun Microsystems.	Chapter 2, "Developing Applications with Standard LDAP APIs"
Oracle Extensions to Standard C, PL/SQL and Java APIs	These APIs provide programmatic interfaces that model various concepts related to identity management.	Chapter 3, "Developing Applications with Oracle Extensions to the Standard APIs"

Table 1–2 (Cont.) Services and APIs for Integrating with Oracle Internet Directory

Service/API	Description	More Information
Oracle Delegated Administration Services	Oracle Delegated Administration Services consists of a self-service console and administrative interfaces. You can modify the administrative interfaces to support third-party applications.	<ul style="list-style-type: none"> Chapter 8, "Integrating with Oracle Delegated Administration Services" The chapter about the delegated administration services framework in <i>Oracle Identity Management Guide to Delegated Administration</i>
Oracle Directory Provisioning Integration Service	You can use the Oracle Provisioning Integration System to provision third-party applications and integrate other provisioning systems.	<ul style="list-style-type: none"> Chapter 6, "Developing Provisioning-Integrated Applications" <i>Oracle Identity Management Integration Guide</i>
Oracle Internet Directory Plug-ins	You can use plug-ins to customize directory behavior in certain deployments.	<ul style="list-style-type: none"> Chapter 7, "Developing Directory Plug-ins" The chapter about plug-ins in <i>Oracle Internet Directory Administrator's Guide</i> Appendix A, "Java Plug-ins for User Provisioning"

Figure 1–2 shows an application leveraging some of the services illustrated in Table 1–2 on page 1-6.

Figure 1–2 An Application Leveraging APIs and Services

As Figure 1–2 shows, the application integrates with Oracle Internet Directory as follows:

- Using PL/SQL, C, or Java APIs, it performs LDAP operations directly against the directory.
- In some cases, it directs users to self-service features of Oracle Delegated Administration Services.
- It is notified of changes to entries for users or groups in Oracle Internet Directory. The Oracle Directory Provisioning Integration Service provides this notification.

Integrating Existing Applications with Oracle Internet Directory

Your enterprise may already have deployed applications that you may have wanted to integrate with the Oracle identity management infrastructure. You can still integrate these applications using the services presented in [Table 1-3](#).

Table 1-3 Services for Modifying Existing Applications

Service	Description	More Information
Automated User Provisioning	You can develop an agent that automatically provisions users when provisioning events occur in the Oracle identity management infrastructure. You use interfaces of the Oracle Directory Provisioning Integration Service to develop this agent.	Chapter 6, "Developing Provisioning-Integrated Applications"
User Authentication Services	If your user interface is based on HTTP, you can integrate it with the Oracle HTTP Server. This enables you to use mod_osso and OracleAS Single Sign-On to protect the application URL.	<i>Oracle Application Server Single Sign-On Administrator's Guide</i>
Centralized User Profile Management	If your user interface is based on HTTP and is integrated with OracleAS Single Sign-On, you can use the Oracle Internet Directory Self-Service Console to manage user profiles centrally. You can tailor the console to the needs of your application.	<ul style="list-style-type: none"> ▪ Chapter 8, "Integrating with Oracle Delegated Administration Services" ▪ The chapter about the delegated administration services framework in <i>Oracle Identity Management Guide to Delegated Administration</i>

Integrating New Applications with Oracle Internet Directory

If you are developing a new application or planning a new release of an existing application, you have many directory integration options at your disposal. [Table 1-4](#) on page 1-9 lists and describes these.

Table 1–4 Application Integration Points

Integration Point	Available Options	More Information
User Authentication Services	<p>If your application is based on J2EE, it can use the JAZN interface to authenticate users. If it relies on OC4J, it can use mod_osso for the same purpose. The second option enables the application to obtain information about the user from HTTP headers.</p> <p>If your application is Web based and standalone, it can still integrate with OracleAS Single Sign-On, then it can still leverage Oracle Application Server Single Sign-On by becoming a partner application using the single sign-on APIs.</p> <p>Finally, if the application provides a non-Web user interface, it can use the Oracle Internet Directory LDAP APIs to integrate users.</p>	<ul style="list-style-type: none"> ■ <i>Oracle Application Server Containers for J2EE User's Guide</i> ■ <i>Oracle Application Server Single Sign-On Administrator's Guide</i> ■ Part II, "Oracle Internet Directory Programming Reference". This section is devoted to the various LDAP APIs.
User Authorization Services	<p>If your application is based on J2EE, it can use the JAZN interface to implement and enforce user authorizations for application resources. The application can define authorizations as groups in Oracle Internet Directory and can then check the authorizations of a user by checking his or her group membership. It can use the Oracle Internet Directory LDAP APIs for this purpose.</p>	<ul style="list-style-type: none"> ■ <i>Oracle Application Server Containers for J2EE User's Guide</i> ■ Part II, "Oracle Internet Directory Programming Reference". This section is devoted to the various LDAP APIs.
Centralized Profile Management	<p>You can define application-specific profiles and user preferences as attributes in Oracle Internet Directory.</p> <p>If your user interface is based on HTTP and is integrated with OracleAS Single Sign-On, you can use the Oracle Internet Directory Self-Service Console to manage user profiles centrally. You can tailor the console to the needs of your application.</p> <p>Additionally, you can use the Oracle Internet Directory LDAP APIs to retrieve user profiles at runtime.</p>	<ul style="list-style-type: none"> ■ The chapter about deployment considerations in <i>Oracle Internet Directory Administrator's Guide</i> ■ Chapter 8, "Integrating with Oracle Delegated Administration Services" ■ <i>Oracle Identity Management Guide to Delegated Administration</i> ■ Part II of this guide, which is devoted to the various LDAP APIs
Automated User Provisioning	<p>If your user interface is based on HTTP and it is integrated with OracleAS Single Sign-On, you can implement automated user provisioning the very first time a user accesses the application.</p> <p>You use the Oracle Directory Provisioning Integration Service to integrate the application with the Oracle identity management infrastructure. Once integrated, the application can provision or deprovision user accounts automatically when an administrator adds, modifies, or deletes an identity.</p>	<p>Chapter 6, "Developing Provisioning-Integrated Applications"</p>

Other Components of Oracle Internet Directory

The SDK is just one component in the directory suite. Here are the others:

- Oracle directory server, LDAP Version 3
- Oracle directory replication server
- Oracle Directory Manager, a Java-based graphical user interface

- Oracle Internet Directory bulk tools
- *Oracle Internet Directory Administrator's Guide*

Developing Applications with Standard LDAP APIs

This chapter takes a high-level look at the operations that the standard LDAP API enables. It explains how to integrate your applications with the API. Before presenting these topics, the chapter revisits the [Lightweight Directory Access Protocol \(LDAP\)](#).

This chapter contains these topics:

- [Sample Code](#)
- [History of LDAP](#)
- [LDAP Models](#)
- [About the Standard LDAP APIs](#)
- [Initializing an LDAP Session](#)
- [Authenticating an LDAP Session](#)
- [Searching the Directory](#)
- [Terminating the Session](#)

Sample Code

Sample code is available at this URL:

http://www.oracle.com/technology/sample_code/

Look for the Oracle Identity Management link under Sample Applications–Oracle Application Server.

History of LDAP

LDAP began as a lightweight front end to the X.500 Directory Access Protocol. LDAP simplifies the X.500 Directory Access Protocol in the following ways:

- It uses TCP/IP connections. These are lightweight compared to the OSI communication stack required by X.500 implementations
- It eliminates little-used and redundant features of the X.500 Directory Access Protocol
- It uses simple formats to represent data elements. These formats are easier to process than the complicated and highly structured representations in X.500.

- It uses a simplified version of the X.500 encoding rules used to transport data over networks.

LDAP Models

LDAP uses four basic models to define its operations:

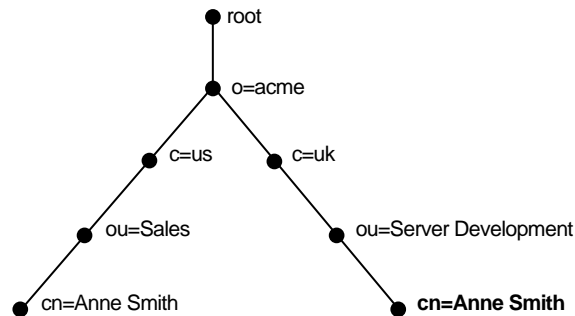
- [Naming Model](#)
- [Information Model](#)
- [Functional Model](#)
- [Security Model](#)

Naming Model

The LDAP naming model enables directory information to be referenced and organized. Each entry in a directory is uniquely identified by a distinguished name (**DN**). The DN tells you exactly where an entry resides in the directory hierarchy. A **directory information tree (DIT)** is used to represent this hierarchy.

[Figure 2-1](#) illustrates the relationship between a distinguished name and a directory information tree.

Figure 2-1 A Directory Information Tree



The DIT in [Figure 2-1](#) shows entries for two employees of Acme Corporation who are both named Anne Smith. It is structured along geographical and organizational lines. The Anne Smith represented by the left branch works in the Sales division in the United States. Her counterpart works in the Server Development division in the United Kingdom.

The Anne Smith represented by the right branch has the common name (**cn**) Anne Smith. She works in an organizational unit (**ou**) named Server Development, in the country (**c**) of United Kingdom of Great Britain and Northern Ireland (**uk**), in the organization (**o**) Acme. The DN for this Anne Smith entry looks like this:

```
cn=Anne Smith,ou=Server Development,c=uk,o=acme
```

Note that the conventional format for a distinguished name places the lowest DIT component at the left. The next highest component follows, on up to the root.

Within a distinguished name, the lowest component is called the **relative distinguished name (RDN)**. In the DN just presented, the RDN is **cn=Anne Smith**. The RDN for the entry immediately above Anne Smith's RDN is **ou=Server Development**. And the RDN for the entry immediately above **ou=Server**

Development is `c=uk`, and so on. A DN is thus a sequence of RDNs separated by commas.

To locate a particular entry within the overall DIT, a client uniquely identifies that entry by using the full DN—not simply the RDN—of that entry. To avoid confusion between the two Anne Smiths in the global organization depicted in [Figure 2-1](#), you use the full DN for each. If there are two employees with the same name in the same organizational unit, you can use other mechanisms. You may, for example, use a unique identification number to identify these employees.

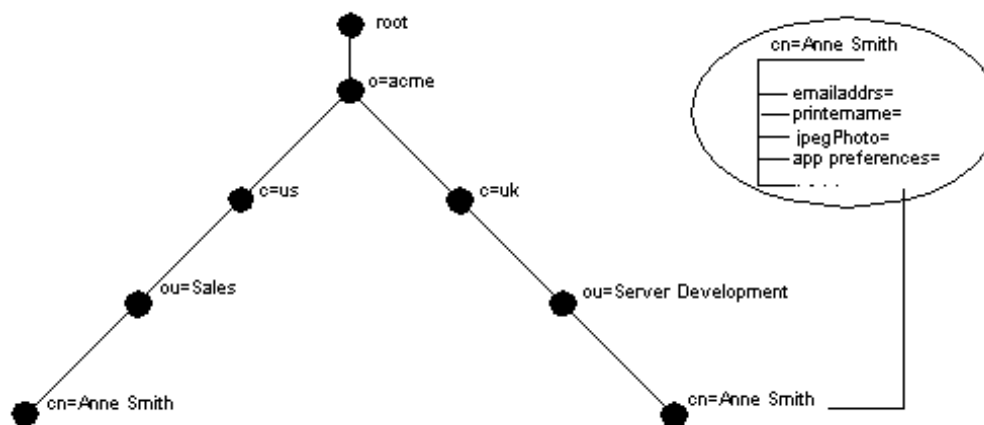
Information Model

The LDAP information model determines the form and character of information in the directory. This model uses the concept of entries as its defining characteristic. In a directory, an **entry** is a collection of information about an object. A telephone directory, for example, contains entries for people. A library card catalog contains entries for books. An online directory may contain entries for employees, conference rooms, e-commerce partners, or shared network resources such as printers.

In a typical telephone directory, a person entry contains an address and a phone number. In an online directory, each of these pieces of information is called an **attribute**. A typical employee entry contains attributes for a job title, an e-mail address, and a phone number.

In [Figure 2-2](#), the entry for Anne Smith in Great Britain (uk) has several attributes. Each provides specific information about her. Those listed in the balloon to the right of the tree are `emailAddr`, `printName`, `jpegPhoto`, and `appPreferences`. Note that the rest of the bullets in [Figure 2-2](#) are also entries with attributes, although these attributes are not shown.

Figure 2-2 Attributes of the Entry for Anne Smith



Each attribute consists of an attribute type and one or more attribute values. The **attribute type** is the kind of information that the attribute contains—`jobTitle`, for instance. The **attribute value** is the actual information. The value for the `jobTitle` attribute, for example, might be `manager`.

Functional Model

The LDAP functional model determines what operations can be performed on directory entries. [Table 2-1](#) on page 2-4 lists and describes the three types of functions:

Table 2–1 LDAP Functions

Function	Description
Search and read	The read operation retrieves the attributes of an entry whose name is known. The list operation enumerates the children of a given entry. The search operation selects entries from a defined area of the tree based on some selection criteria known as a search filter. For each matching entry, a requested set of attributes (with or without values) is returned. The searched entries can span a single entry, an entry's children, or an entire subtree. Alias entries can be followed automatically during a search, even if they cross server boundaries. An abandon operation is also defined, allowing an operation in progress to be canceled.
Modify	This category defines four operations that modify the directory: <ul style="list-style-type: none"> ▪ Modify—change existing entries. You can add and delete values. ▪ Add—insert entries into the directory ▪ Delete—remove entries from the directory ▪ Modify RDN—change the name of an entry
Authenticate	This category defines a bind operation. A bind enables a client to initiate a session and prove its identity to the directory. Oracle Internet Directory supports several authentication methods, from simple clear-text passwords to public keys. The unbind operation is used to terminate a directory session.

Security Model

The LDAP security model enables directory information to be secured. This model has several parts:

- **Authentication**
Ensuring that the identities of users, hosts, and clients are correctly validated
- **Access Control and Authorization**
Ensuring that a user reads or updates only the information for which that user has privileges
- **Data Integrity**: Ensuring that data is not modified during transmission
- **Data Privacy**
Ensuring that data is not disclosed during transmission
- **Password Policies**
Setting rules that govern how passwords are used

Authentication

Authentication is the process by which the directory server establishes the identity of the user connecting to the directory. Directory authentication occurs when an LDAP bind operation establishes an LDAP session. Every session has an associated user identity, also referred to as an authorization ID.

Oracle Internet Directory provides three authentication options: anonymous, simple, and SSL.

Anonymous Authentication If your directory is available to everyone, users may log in anonymously. In **anonymous authentication**, users leave the user name and password fields blank when they log in. They then exercise whatever privileges are specified for anonymous users.

Simple Authentication In **simple authentication**, the client uses an unencrypted DN and password to identify itself to the server. The server verifies that the client's DN and password match the DN and password stored in the directory.

Authentication Using Secure Sockets Layer (SSL) **Secure Sockets Layer (SSL)** is an industry standard protocol for securing network connections. It uses a **certificate** exchange to authenticate users. These certificates are verified by trusted certificate authorities. A certificate ensures that an entity's identity information is correct. An entity can be an end user, a database, an administrator, a client, or a server. A **Certificate Authority (CA)** is an application that creates public key certificates that are given a high level of trust by all parties involved.

You can use SSL in one of the three authentication modes presented in [Table 2–2](#).

Table 2–2 SSL Authentication Modes

SSL Mode	Description
No authentication	Neither the client nor the server authenticates itself to the other. No certificates are sent or exchanged. In this case, only SSL encryption and decryption are used.
One-way authentication	Only the directory server authenticates itself to the client. The directory server sends the client a certificate verifying that the server is authentic.
Two-way authentication	Both client and server authenticate themselves to each other, exchanging certificates.

In an Oracle Internet Directory environment, SSL authentication between a client and a directory server involves three basic steps:

1. The user initiates an LDAP connection to the directory server by using SSL on an SSL port. The default SSL port is 636.
2. SSL performs the handshake between the client and the directory server.
3. If the handshake is successful, the directory server verifies that the user has the appropriate authorization to access the directory.

See Also: *Oracle Advanced Security Administrator's Guide* for more information about SSL

Access Control and Authorization

The authorization process ensures that a user reads or updates only the information for which he or she has privileges. The directory server ensures that the user—identified by the authorization ID associated with the session—has the requisite permissions to perform a given directory operation. Absent these permissions, the operation is disallowed.

The mechanism that the directory server uses to ensure that the proper authorizations are in place is called access control. And an **access control item (ACI)** is the directory metadata that captures the administrative policies relating to access control.

An ACI is stored in Oracle Internet Directory as user-modifiable operational attributes. Typically a whole list of these ACI attribute values is associated with a directory object.

This list is called an **access control list (ACL)**. The attribute values on that list govern the access policies for the directory object.

ACIs are stored as text strings in the directory. These strings must conform to a well-defined format. Each valid value of an ACI attribute represents a distinct access control policy. These individual policy components are referred to as ACI Directives or ACIs and their format is called the ACI Directive format.

Access control policies can be prescriptive: their security directives can be set to apply downward to all entries at lower positions in the **directory information tree (DIT)**. The point from which an access control policy applies is called an **access control policy point (ACP)**.

Data Integrity

Oracle Internet Directory uses SSL to ensure that data is not modified, deleted, or replayed during transmission. This feature uses cryptographic checksums to generate a secure message digest. The checksums are created using either the **MD5** algorithm or the **Secure Hash Algorithm (SHA)**. The message digest is included in each network packet.

Data Privacy

Oracle Internet Directory uses **public key encryption** over SSL to ensure that data is not disclosed during transmission. In public-key encryption, the sender of a message encrypts the message with the public key of the recipient. Upon delivery, the recipient decrypts the message using his or her private key. The directory supports two levels of encryption:

- **DES40**

The DES40 algorithm, available internationally, is a **DES** variant in which the secret key is preprocessed to provide forty effective **key** bits. It is designed for use by customers outside the USA and Canada who want to use a DES-based encryption algorithm.

- **RC4_40**

Oracle is licensed to export the RC4 data encryption algorithm with a 40-bit key size to virtually all destinations where Oracle products are available. This makes it possible for international corporations to safeguard their entire operations with fast cryptography.

Password Policies

A password policy is a set of rules that govern how passwords are used. When a user attempts to bind to the directory, the directory server uses the password policy to ensure that the password provided meets the various requirements set in that policy.

When you establish a password policy, you set the following types of rules, to mention just a few:

- The maximum length of time a given password is valid
- The minimum number of characters a password must contain
- The ability of users to change their passwords

About the Standard LDAP APIs

The standard LDAP APIs enable you to perform the fundamental LDAP operations described in "LDAP Models". These APIs are available in C, PL/SQL, and Java. The first two are part of the directory SDK. The last is part of the JNDI package provided by Sun Microsystems. All three use TCP/IP connections. They are based on LDAP Version 3, and they support SSL connections to Oracle Internet Directory.

This section contains these topics:

- [API Usage Model](#)
- [Getting Started with the C API](#)
- [Getting Started with the Java API](#)
- [Getting Started with the DBMS_LDAP Package](#)

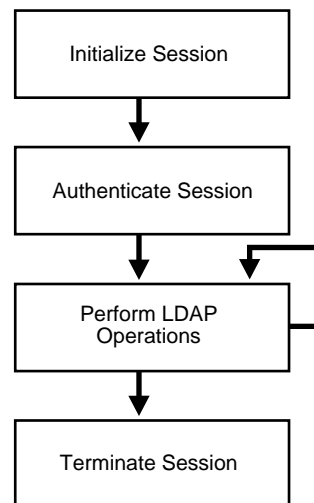
API Usage Model

Typically, an application uses the functions in the API in four steps:

1. Initialize the library and obtain an LDAP session handle.
2. Authenticate to the LDAP server if necessary.
3. Perform some LDAP operations and obtain results and errors, if any.
4. Close the session.

[Figure 2-3](#) illustrates these steps.

Figure 2-3 Steps in Typical DBMS_LDAP Usage



Getting Started with the C API

When you build applications with the C API, you must include the header file `ldap.h`, located at `$ORACLE_HOME/ldap/public`. In addition, you must dynamically link to the library located at `$ORACLE_HOME/lib/libclntsh.so.10.1`.

See Also: "[Sample C API Usage](#)" on page 11-40 to learn how to use the SSL and non-SSL modes

Getting Started with the DBMS_LDAP Package

The `DBMS_LDAP` package enables PL/SQL applications to access data located in enterprise-wide LDAP servers. The names and syntax of the function calls are similar to those of the C API. These functions comply with current recommendations of the [Internet Engineering Task Force \(IETF\)](#) for the C API. Note though that the PL/SQL API contains only a subset of the functions available in the C API. Most notably, only synchronous calls to the LDAP server are available in the PL/SQL API.

To begin using the PL/SQL LDAP API, use this command sequence to load `DBMS_LDAP` into the database:

1. Log in to the database, using `SQL*Plus`. Run the tool in the Oracle home in which your database is present. Connect as `SYSDBA`.

```
SQL> CONNECT / AS SYSDBA
```

2. Load the API into the database, using this command:

```
SQL> @?/rdbs/admin/catldap.sql
```

Getting Started with the Java API

Java developers can use the Java Naming and Directory Interface (JNDI) from Sun Microsystems to gain access to information in Oracle Internet Directory. The JNDI is found at this link:

<http://java.sun.com/products/jndi>

Although no Java APIs are provided in this chapter, the section immediately following, "[Initializing the Session by Using JNDI](#)", shows you how to use wrapper methods for the Sun JNDI to establish a basic connection.

Initializing an LDAP Session

All LDAP operations based on the C API require clients to establish an LDAP session with the LDAP server. For LDAP operations based on the PL/SQL API, a database session must first initialize and open an LDAP session. Most Java operations require a Java Naming and Directory Interface (JNDI) connection. The `oracle.ldap.util.jndi` package, provided here, simplifies the work involved in achieving this connection.

The section contains the following topics:

- [Initializing the Session by Using the C API](#)
- [Initializing the Session by Using DBMS_LDAP](#)
- [Initializing the Session by Using JNDI](#)

Initializing the Session by Using the C API

The C function `ldap_init()` initializes a session with an LDAP server. The server is not actually contacted until an operation is performed that requires it, allowing various options to be set after initialization.

`ldap_init` has the following syntax:

```
LDAP *ldap_init
(
const char      *hostname,
int             portno
```

);

Table 2–3 lists and defines the function parameters.

Table 2–3 Parameters for `ldap_init()`

Parameter	Description
hostname	<p>Contains a space-separated list of directory host names or IP addresses represented by dotted strings. You can pair each host name with a port number as long as you use a colon to separate the two.</p> <p>The hosts are tried in the order listed until a successful connection is made.</p> <p>Note: A suitable representation for including a literal IPv6[10] address in the host name parameter is desired, but has not yet been determined or implemented in practice.</p>
portno	<p>Contains the TCP port number of the directory you would like to connect to. The default LDAP port of 389 can be obtained by supplying the constant <code>LDAP_PORT</code>. If a host includes a port number, this parameter is ignored.</p>

`ldap_init()` and `ldap_open()` both return a session handle, or pointer, to an opaque structure that must be passed to subsequent calls to the session. These routines return `NULL` if the session cannot be initialized. You can check the error reporting mechanism for your operating system to determine why the call failed.

Initializing the Session by Using `DBMS_LDAP`

In the PL/SQL API, the function `DBMS_LDAP.init()` initiates an LDAP session. This function has the following syntax:

```
FUNCTION init (hostname IN VARCHAR2, portnum IN PLS_INTEGER )
RETURN SESSION;
```

The function `init` requires a valid host name and port number to establish an LDAP session. It allocates a data structure for this purpose and returns a handle of the type `DBMS_LDAP.SESSION` to the caller. The handle returned from the call should be used in all subsequent LDAP operations defined by `DBMS_LDAP` for the session. The API uses these session handles to maintain state about open connections, outstanding requests, and other information.

A single database session can obtain as many LDAP sessions as required, although the number of simultaneous active connections is limited to 64. One database session typically has multiple LDAP sessions when data must be obtained from multiple servers simultaneously or when open sessions that use multiple LDAP identities are required.

Note: The handles returned from calls to `DBMS_LDAP.init()` are dynamic constructs. They do not persist across multiple database sessions. Attempting to store their values in a persistent form, and to reuse stored values at a later stage, can yield unpredictable results.

Initializing the Session by Using JNDI

The `oracle.ldap.util.jndi` package supports basic connections by providing wrapper methods for the JNDI implementation from Sun Microsystems. If you want to use the JNDI to establish a connection, see the following link:

<http://java.sun.com/products/jndi>

Here is an implementation of `oracle.ldap.util.jndi` that establishes a non-SSL connection:

```
import oracle.ldap.util.jndi
import javax.naming.*;

public static void main(String args[])
{
    try{
        InitialDirContext ctx = ConnectionUtil.getDefaultDirCtx(args[0], // host
                                                                args[1], // port
                                                                args[2], // DN
                                                                args[3]; // password)

        // Do work
    }
    catch(NamingException ne)
    {
        // javax.naming.NamingException is thrown when an error occurs
    }
}
```

Note:

- `DN` and `password` represent the bind DN and password. For anonymous binds, set these to "".
 - You can use `ConnectionUtil.getSSLDirCtx()` to establish a no-authentication SSL connection.
-
-

Authenticating an LDAP Session

Individuals or applications seeking to perform operations against an LDAP server must first be authenticated. If the `dn` and `passwd` parameters of these entities are null, the LDAP server assigns a special identity, called anonymous, to these users. Typically, the anonymous user is the least privileged user of the directory.

Once a bind operation is complete, the directory server remembers the new identity until another bind occurs or the LDAP session terminates (`unbind_s`). The LDAP server uses the identity to enforce the security model specified by the enterprise in which it is deployed. The identity helps the LDAP server determine whether the user or application identified has sufficient privileges to perform search, update, or compare operations in the directory.

Note that the password for the bind operation is sent over the network in clear text. If your network is not secure, consider using SSL for authentication and other LDAP operations that involve data transfer.

This section contains these topics:

- [Authenticating an LDAP Session by Using the C API](#)
- [Authenticating an LDAP Session by Using DBMS_LDAP](#)

Authenticating an LDAP Session by Using the C API

The C function `ldap_simple_bind_s()` enables users and applications to authenticate to the directory server using a DN and password.

The function `ldap_simple_bind_s()` has this syntax:

```
int ldap_simple_bind_s
(
LDAP* ld,
char* dn,
char* passwd
);
```

Table 2–4 lists and describes the parameters for this function.

Table 2–4 Arguments for `ldap_simple_bind_s()`

Argument	Description
<code>ld</code>	A valid LDAP session handle
<code>dn</code>	The identity that the application uses for authentication
<code>passwd</code>	The password for the authentication identity

If the `dn` and `passwd` parameters are `NULL`, the LDAP server assigns a special identity, called anonymous, to the user or application.

Authenticating an LDAP Session by Using `DBMS_LDAP`

The PL/SQL function `simple_bind_s` enables users and applications to use a DN and password to authenticate to the directory. `simple_bind_s` has this syntax:

```
FUNCTION simple_bind_s ( ld IN SESSION, dn IN VARCHAR2, passwd IN VARCHAR2)
RETURN PLS_INTEGER;
```

Note that this function requires as its first parameter the LDAP session handle obtained from `init`.

The following PL/SQL code snippet shows how the PL/SQL initialization and authentication functions just described might be implemented.

```
DECLARE
retval PLS_INTEGER;
my_session DBMS_LDAP.session;

BEGIN
retval := -1;
-- Initialize the LDAP session
my_session := DBMS_LDAP.init('yow.acme.com', 389);
-- Authenticate to the directory
retval := DBMS_LDAP.simple_bind_s(my_session, 'cn=orcladmin',
'welcome');
```

In the previous example, an LDAP session is initialized on the LDAP server `yow.acme.com`. This server listens for requests at TCP/IP port number 389. The identity `cn=orcladmin`, whose password is `welcome`, is then authenticated. Once authentication is complete, regular LDAP operations can begin.

Searching the Directory

Searches are the most common LDAP operations. Applications can use complex search criteria to select and retrieve entries from the directory.

This section contains these topics:

- [Program Flow for Search Operations](#)
- [Search Scope](#)
- [Filters](#)
- [Searching the Directory by Using the C API](#)
- [Searching the Directory by Using DBMS_LDAP](#)

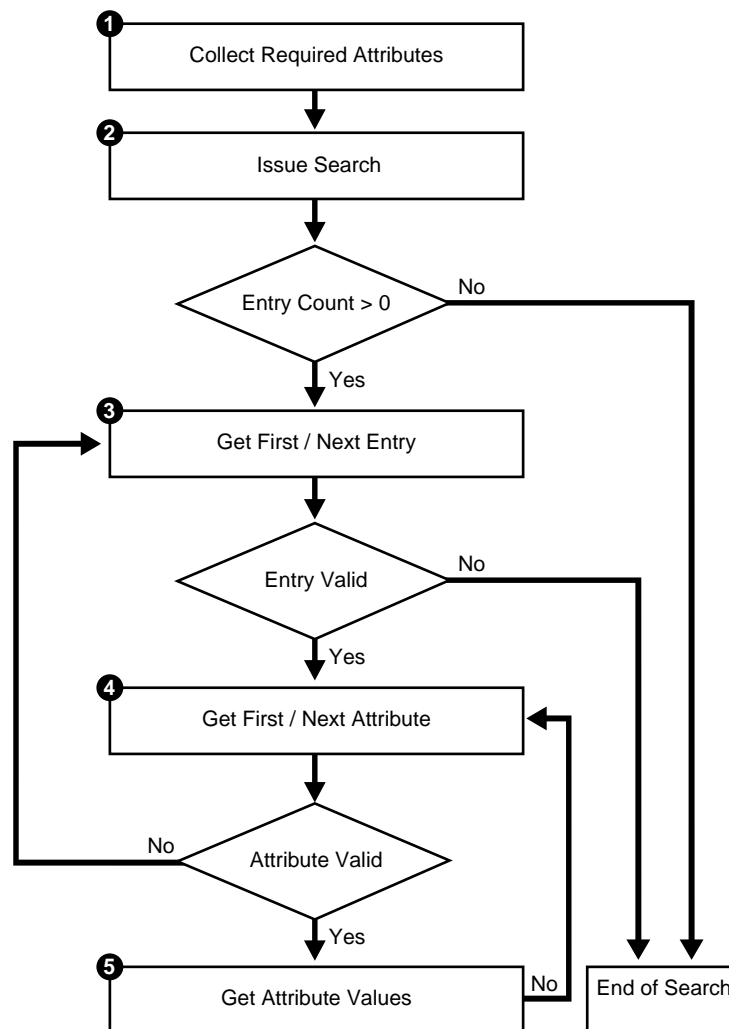
Note: This release of the `DBMS_LDAP` API provides only synchronous search capability. This means that the caller of the search functions is blocked until the LDAP server returns the entire result set.

Program Flow for Search Operations

The programming required to initiate a typical search operation and retrieve results can be broken down into the following steps:

1. Decide what attributes must be returned; then place them into an array.
2. Initiate the search, using the scope options and filters of your choice.
3. Obtain an entry from result set.
4. Obtain an attribute from the entry obtained in step 3.
5. Obtain the values of the attributes obtained in step 4; then copy these values into local variables.
6. Repeat step 4 until all attributes of the entry are examined.
7. Repeat Step 3 until there are no more entries

[Figure 2-4](#) on page 2-13 uses a flow chart to represent these steps.

Figure 2-4 Flow of Search-Related Operations

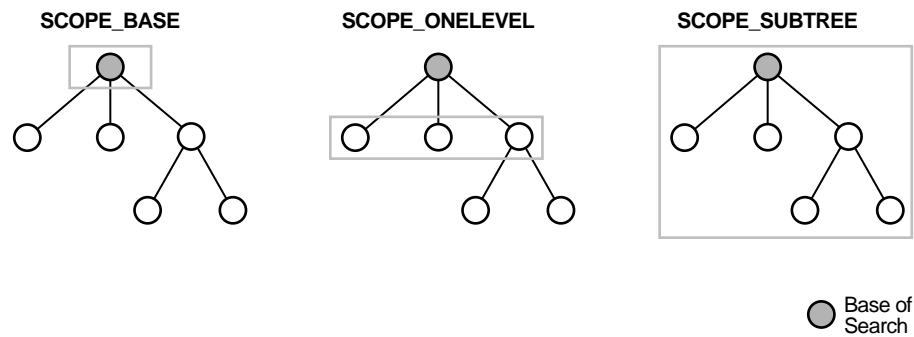
Search Scope

The scope of a search determines how many entries the directory server examines relative to the search base. You can choose one of the three options described in [Table 2-5](#) and illustrated in [Figure 2-5](#) on page 2-14.

Table 2-5 Options for `search_s()` or `search_st()` Functions

Option	Description
SCOPE_BASE	The directory server looks only for the entry corresponding to the search base.
SCOPE_ONELEVEL	The directory server confines its search to the entries that are the immediate children of the search base entry.
SCOPE_SUBTREE	The directory server looks at the search base entry and the entire subtree beneath it.

Figure 2-5 The Three Scope Options



In Figure 2-5, the search base is the shaded circle. The shaded rectangle identifies the entries that are searched.

Filters

A search filter is an expression that enables you to confine your search to certain types of entries. The search filter required by the `search_s()` and `search_st()` functions follows the string format defined in RFC 1960 of the Internet Engineering Task Force (IETF). As Table 2-6 shows, there are six kinds of search filters. These are entered in the format *attribute operator value*.

Table 2-6 Search Filters

Filter Type	Format	Example	Matches
Equality	<code>(att=value)</code>	<code>(sn=Keaton)</code>	Surnames exactly equal to Keaton.
Approximate	<code>(att~=value)</code>	<code>(sn~=Ketan)</code>	Surnames approximately equal to Ketan.
Substring	<code>(attr=[leading]*[any]*[trailing])</code>	<code>(sn=*keaton*)</code>	Surnames containing the string keaton.
		<code>(sn=keaton*)</code>	Surnames starting with keaton.
		<code>(sn=*keaton)</code>	Surnames ending with keaton.
		<code>(sn=ke*at*on)</code>	Surnames starting with ke, containing at and ending with on.
Greater than or equal	<code>attr>=value</code>	<code>(sn>=Keaton)</code>	Surnames lexicographically greater than or equal to Keaton.
Less than or equal	<code>(attr<=value)</code>	<code>(sn<=Keaton)</code>	Surnames lexicographically less than or equal to Keaton.
Presence	<code>(attr=*)</code>	<code>(sn=*)</code>	All entries having the sn attribute.

You can use boolean operators and prefix notation to combine these filters to form more complex filters. Table 2-7 on page 2-15 provides examples. In these examples, the

& character represents AND, the | character represents OR, and the ! character represents NOT.

Table 2–7 Boolean Operators

Filter Type	Format	Example	Matches
AND	<code>(&(filter1)(filter2)). . .)</code>	<code>(&(sn=keaton)(objectclass=inetOrgPerson))</code>	Entries with surname of Keaton and object class of InetOrgPerson.
OR	<code>((filter1)(filter2)). . .)</code>	<code>((sn~=ketan)(cn=*keaton))</code>	Entries with surname approximately equal to ketan or common name ending in keaton.
NOT	<code>(!(filter))</code>	<code>(!(mail=*))</code>	Entries without a mail attribute.

The complex filters in [Table 2–7](#) can themselves be combined to create even more complex, nested filters.

Searching the Directory by Using the C API

The C function `ldap_search_s()` performs a synchronous search of the directory.

The syntax for `ldap_search_s()` looks like this:

```
int ldap_search_s
(
LDAP*      ld,
char*      base,
int        scope,
char*      filter,
int        attrsonly,
LDAPMessage** res
);
```

`ldap_search_s` works with several supporting functions to refine the search. The steps that follow show how all of these C functions fit into the program flow of a search operation. [Chapter 11, "C API Reference"](#), examines all of these functions in depth.

1. Decide what attributes must be returned; then place them into an array of strings. The array must be null terminated.
2. Initiate the search, using `ldap_search_s()`. Refine your search with scope options and filters.
3. Obtain an entry from the result set, using either the `ldap_first_entry()` function or the `ldap_next_entry()` function.
4. Obtain an attribute from the entry obtained in step 3. Use either the `ldap_first_attribute()` function or the `ldap_next_attribute()` function for this purpose.
5. Obtain all the values for the attribute obtained in step 4; then copy these values into local variables. Use the `ldap_get_values()` function or the `ldap_get_values_len()` function for this purpose.
6. Repeat step 4 until all attributes of the entry are examined.

- Repeat step 3 until there are no more entries.

Table 2–8 Arguments for `ldap_search_s()`

Argument	Description
<code>ld</code>	A valid LDAP session handle
<code>base</code>	The DN of the search base.
<code>scope</code>	The breadth and depth of the DIT to be searched.
<code>filter</code>	The filter used to select entries of interest.
<code>attrs</code>	The attributes of interest in the entries returned.
<code>attrso</code>	If set to 1, only returns attributes.
<code>res</code>	This argument returns the search results.

Searching the Directory by Using `DBMS_LDAP`

You use the function `DBMS_LDAP.search_s()` to perform directory searches if you use the PL/SQL API.

Here is the syntax for `DBMS_LDAP.search_s()`:

```
FUNCTION search_s
(
  ld      IN  SESSION,
  base    IN  VARCHAR2,
  scope   IN  PLS_INTEGER,
  filter  IN  VARCHAR2,
  attrs   IN  STRING_COLLECTION,
  attronly IN PLS_INTEGER,
  res     OUT MESSAGE
)
RETURN PLS_INTEGER;
```

The function takes the arguments listed and described in [Table 2–9](#) on page 2-16.

Table 2–9 Arguments for `DBMS_LDAP.search_s()` and `DBMS_LDAP.search_st()`

Argument	Description
<code>ld</code>	A valid session handle
<code>base</code>	The DN of the base entry in the LDAP server where search should start
<code>scope</code>	The breadth and depth of the DIT that needs to be searched
<code>filter</code>	The filter used to select entries of interest
<code>attrs</code>	The attributes of interest in the entries returned
<code>attronly</code>	If set to 1, only returns the attributes
<code>res</code>	An OUT parameter that returns the result set for further processing

`search_s` works with several supporting functions to refine the search. The steps that follow show how all of these PL/SQL functions fit into the program flow of a search operation.

- Decide what attributes need to be returned; then place them into the `DBMS_LDAP.STRING_COLLECTION` data-type.

2. Perform the search, using either `DBMS_LDAP.search_s()` or `DBMS_LDAP.search_st()`. Refine your search with scope options and filters.
3. Obtain an entry from the result set, using either `DBMS_LDAP.first_entry()` or `DBMS_LDAP.next_entry()`.
4. Obtain an attribute from the entry obtained in step 3. Use either `DBMS_LDAP.first_attribute()` or `DBMS_LDAP.next_attribute()` for this purpose.
5. Obtain all the values for the attribute obtained in step 4; then copy these values into local variables. Use either `DBMS_LDAP.get_values()` or `DBMS_LDAP.get_values_len()` for this purpose.
6. Repeat step 4 until all attributes of the entry are examined.
7. Repeat step 3 until there are no more entries.

Terminating the Session

This section contains these topics:

- [Terminating the Session by Using the C API](#)
- [Terminating the Session by Using DBMS_LDAP](#)

Terminating the Session by Using the C API

Once an LDAP session handle is obtained and all directory-related work is complete, the LDAP session must be destroyed. In the C API, the `ldap_unbind_s()` function is used for this purpose.

`ldap_unbind_s()` has this syntax:

```
int ldap_unbind_s
(
LDAP* ld
);
```

A successful call to `ldap_unbind_s()` closes the TCP/IP connection to the directory. It de-allocates system resources consumed by the LDAP session. Finally it returns the integer `LDAP_SUCCESS` to its callers. Once `ldap_unbind_s()` is invoked, no other LDAP operations are possible. A new session must be started with `ldap_init()`.

Terminating the Session by Using DBMS_LDAP

The `DBMS_LDAP.unbind_s()` function destroys an LDAP session if the PL/SQL API is used. `unbind_s` has the following syntax:

```
FUNCTION unbind_s (ld IN SESSION ) RETURN PLS_INTEGER;
```

`unbind_s` closes the TCP/IP connection to the directory. It de-allocates system resources consumed by the LDAP session. Finally it returns the integer `DBMS_LDAP.SUCCESS` to its callers. Once the `unbind_s` is invoked, no other LDAP operations are possible. A new session must be initiated with the `init` function.

Developing Applications with Oracle Extensions to the Standard APIs

This chapter introduces the Oracle extensions to the Java and PL/SQL LDAP APIs. Chapter 4 explains how the Java extensions are used. Chapter 5 is about the PL/SQL extensions. Oracle does not support extensions to the C API.

This chapter contains these topics:

- [Sample Code](#)
- [Using Oracle Extensions to the Standard APIs](#)
- [Creating an Application Identity in the Directory](#)
- [Managing Users](#)
- [Managing Groups](#)
- [Managing Realms](#)
- [Discovering a Directory Server](#)
- [SASL Authentication](#)
- [Proxying on Behalf of End Users](#)
- [Creating Dynamic Password Verifiers](#)

Sample Code

Sample code is available at this URL:

http://www.oracle.com/technology/sample_code/

Look for the Oracle Identity Management link under Sample Applications–Oracle Application Server.

Using Oracle Extensions to the Standard APIs

The APIs that Oracle has added to the existing APIs fulfill these functions:

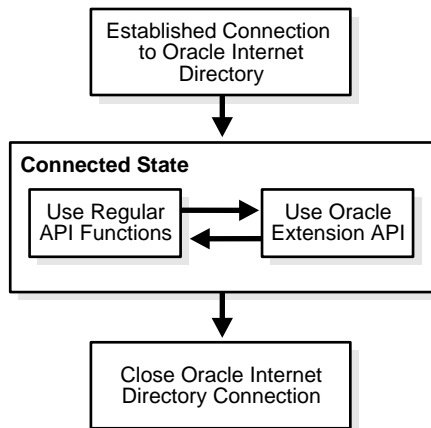
- User management
 - Applications can set or retrieve various user properties
- Group management
 - Applications can query group properties
- Realm management

- Applications can set or retrieve properties about identity management realms
- Server discovery management
 - Applications can locate a directory server in the Domain Name System (DNS)
- SASL management
 - Applications can authenticate to the directory using SASL Digest-MD5

Subsequent sections examine each of these functions in detail. Note that applications must use the underlying APIs for such common tasks as establishing and closing connections and looking up directory entries not searchable with the API extensions.

Figure 3-1 shows what program flow looks like when the API extensions are used.

Figure 3-1 Programmatic Flow for API Extensions



As Figure 3-1 shows, an application first establishes a connection to Oracle Internet Directory. It can then use the standard API functions and the API extensions interchangeably.

Creating an Application Identity in the Directory

Before an application can use the LDAP APIs and their extensions, it must establish an LDAP connection. Once it establishes a connection, it must have permission to perform operations. But neither task can be completed if the application lacks an identity in the directory.

Creating an Application Identity

Creating an application identity in the directory is relatively simple. Such an entry requires only two object classes: `orclApplicationEntity` and `top`. You can use either Oracle Directory Manager or an LDIF file to create the entry. In LDIF notation, the entry looks like this:

```

dn: orclapplicationcommonname=application_name
changetype: add
objectclass: top
objectclass: orclApplicationEntity
userpassword: password
    
```

The value provided for `userpassword` is the value that the application uses to bind to the directory.

Assigning Privileges to an Application Identity

To learn about the privileges available to an application, see the chapter about delegating privileges for an Oracle technology deployment in *Oracle Internet Directory Administrator's Guide*. After identifying the right set of privileges, add the application entity DN to the appropriate directory groups. The link just provided explains how to perform this task using either Oracle Directory Manager or the `ldapmodify` command.

Managing Users

This section describes user management features of the LDAP APIs.

Directory-enabled applications need to perform the following operations:

- Retrieve properties of user entries
These properties are stored as attributes of the user entry itself—in the same way, for example, that a surname or a home address is stored.
- Retrieve extended user preferences
These preferences apply to a user but are stored in a DIT different from the DIT containing user entries. Extended user preferences are either user properties common to all applications or user properties specific to an application. Those of the first type are stored in a common location in the Oracle Context. Those of the second type are stored in the application-specific DIT.
- Query the group membership of a user
- Authenticate a user given a simple name and credential
Typically an application uses a fully qualified DN, GUID, or simple user name to identify a user. In a hosted environment, the application may use both a user name and a realm name for identification.

Managing Groups

Groups are modeled in Oracle Internet Directory as a collection of distinguished names. Directory-enabled applications must access Oracle Internet Directory to obtain the properties of a group and to verify that a given user is a member of that group.

A group is typically identified by one of the following:

- A fully qualified LDAP distinguished name
- A global unique identifier
- A simple group name along with a subscriber name

Managing Realms

An identity management realm is an entity or organization that subscribes to the services offered in the Oracle product stack. Directory-enabled applications must access Oracle Internet Directory to obtain realm properties such as user search base or password policy.

A realm is typically identified by one of the following:

- A fully qualified LDAP distinguished name
- A global unique identifier

- A simple enterprise name

Discovering a Directory Server

Directory server discovery (DSD) enables automatic discovery of the Oracle directory server by directory clients. It enables deployments to manage the directory host name and port number information in the central DNS server. All directory clients perform a DNS query at runtime and connect to the directory server. Directory server location information is stored in a DNS service location record (SRV).

An SRV contains:

- The DNS name of the server providing LDAP service
- The port number of the corresponding port
- Any parameters that enable the client to choose an appropriate server from multiple servers

DSD also allows clients to discover the directory host name information from the `ldap.ora` file itself.

This section contains these topics:

- [Benefits of Oracle Internet Directory Discovery Interfaces](#)
- [Usage Model for Discovery Interfaces](#)
- [Determining Server Name and Port Number From DNS](#)
- [Environment Variables for DNS Server Discovery](#)
- [Programming Interfaces for DNS Server Discovery](#)

See Also:

- "Discovering LDAP Services with DNS" by Michael P. Armijo at this URL:
<http://www.ietf.org/>
- "A DNS RR for specifying the location of services (DNS SRV)", Internet RFC 2782 at the same URL.

Benefits of Oracle Internet Directory Discovery Interfaces

Typically, the LDAP host name and port information is provided statically in a file called `ldap.ora` which is located on the client in `$ORACLE_HOME/network/admin`. For large deployments with many clients, this information becomes very cumbersome to manage. For example, each time the host name or port number of a directory server is changed, the `ldap.ora` file on each client must be modified.

Directory server discovery eliminates the need to manage the host name and port number in the `ldap.ora` file. Because the host name information resides on one central DNS server, the information must be updated only once. All clients can then discover the new host name information dynamically from the DNS when they connect to it.

DSD provides a single interface to obtain directory server information without regard to the mechanism or standard used to obtain it. Currently, Oracle directory server information can be obtained either from DNS or from `ldap.ora` using a single interface.

Usage Model for Discovery Interfaces

The first step in discovering host name information is to create a discovery handle. A discovery handle specifies the source from which host name information will be discovered. In case of the Java API, the discovery handle is created by creating an instance of the `oracle.ldap.util.discovery.DiscoveryHelper` class.

```
DiscoveryHelper disco = new DiscoveryHelper(DiscoveryHelper.DNS_DISCOVER);
```

The argument `DiscoveryHelper.DNS_DISCOVER` specifies the source. In this case the source is DNS.

Each source may require some inputs to be specified for discovery of host name information. In the case of DNS these inputs are:

- domain name
- discover method
- SSL mode

Detailed explanation of these options is given in ["Determining Server Name and Port Number From DNS"](#).

```
// Set the property for the DNS_DN
disco.setProperty(DiscoveryHelper.DNS_DN, "dc=us,dc=fiction,dc=com");
// Set the property for the DNS_DISCOVER_METHOD
disco.setProperty(DiscoveryHelper.DNS_DISCOVER_METHOD
    ,DiscoveryHelper.USE_INPUT_DN_METHOD);
// Set the property for the SSLMODE
disco.setProperty(DiscoveryHelper.SSLMODE, "0");
```

Now the information can be discovered.

```
// Call the discover method
disco.discover(reshdl);
```

The discovered information is returned in a result handle (`reshdl`). Now the results can be extracted from the result handle.

```
ArrayList result =
(ArrayList)reshdl.get(DiscoveryHelper.DIR_SERVERS);
if (result != null)
{
    if (result.size() == 0) return;
    System.out.println("The hostnames are :-");
    for (int i = 0; i < result.size(); i++)
    {
        String host = (String)result.get(i);
        System.out.println((i+1)+".'"+host+"'");
    }
}
```

Determining Server Name and Port Number From DNS

Determining a host name and port number from a DNS lookup involves obtaining a domain and then searching for SRV resource records based on that domain. If there is more than one SRV resource record, they are sorted by weight and priority. The SRV resource records contain host names and port numbers required for connection. This information is retrieved from the resource records and returned to the user.

There are three approaches for determining the domain name required for lookup:

- Mapping the distinguished name (DN) of the naming context
- Using the domain component of local machine
- Looking up the default SRV record in the DNS

Mapping the DN of the Naming Context

The first approach is to map the distinguished name (DN) of naming context into domain name using the algorithm given here.

The output domain name is initially empty. The DN is processed sequentially from right to left. An RDN is able to be converted if it meets the following conditions:

- It consists of a single attribute type and value
- The attribute type is `dc`
- The attribute value is non-null

If the RDN can be converted, then the attribute value is used as a domain name component (label).

The first such value becomes the rightmost, and the most significant, domain name component. Successive converted RDN values extend to the left. If an RDN cannot be converted, then processing stops. If the output domain name is empty when processing stops, then the DN cannot be converted into a domain name.

For the DN `cn=John Doe,ou=accounting,dc=example,dc=net`, the client converts the `dc` components into the DNS name `example.net`.

Search by Domain Component of Local Machine

Sometimes a DN cannot be mapped to a domain name. For example, the DN `o=Oracle IDC,Bangalore` cannot be mapped to a domain name. In this case, the second approach uses the domain component of the local machine on which the client is running. For example, if the client machine domain name is `mc1.acme.com`, the domain name for the lookup is `acme.com`.

Search by Default SRV Record in DNS

The third approach looks for a default SRV record in the DNS. This record points to the default server in the deployment. The domain component for this default record is `_default`.

Once the domain name has been determined, it is used to send a query to DNS. The DNS is queried for SRV records specified in Oracle Internet Directory-specific format. For example, if the domain name obtained is `example.net`, the query for non-SSL LDAP servers is for SRV resource records having the owner name `_ldap._tcp._oid.example.net`.

It is possible that no SRV resource records are returned from the DNS. In such a case the DNS lookup is performed for the SRV resource records specified in standard format. For example, the owner name would be `_ldap._tcp.example.net`.

See Also: The chapter about directory administration in *Oracle Internet Directory Administrator's Guide*

The result of the query is a set of SRV records. These records are then sorted and the host information is extracted from them. This information is then returned to the user.

Note: The approaches mentioned here can also be tried in succession, stopping when the query lookup of DNS is successful. Try the approaches in the order as described in this section. DNS is queried only for SRV records in Oracle Internet Directory-specific format. If none of the approaches is successful, then all the approaches are tried again, but this time DNS is queried for SRV records in standard format.

Environment Variables for DNS Server Discovery

The following environment variables override default behavior for discovering a DNS server.

Table 3–1 *Environment Variables for DNS Discovery*

Environment Variable	Description
ORA_LDAP_DNS	IP address of the DNS server containing the SRV records. If the variable is not defined, then the DNS server address is obtained from the host machine.
ORA_LDAP_DNSPORT	Port number on which the DNS server listens for queries. If the variable is not defined, then the DNS server is assumed to be listening at standard port number 53.
ORA_LDAP_DOMAIN	Domain of the host machine. If the variable is not defined, then the domain is obtained from the host machine itself.

Programming Interfaces for DNS Server Discovery

The programming interface provided is a single interface to discover directory server information without regard to the mechanism or standard used to obtain it. Information can be discovered from various sources. Each source can use its own mechanism to discover the information. For example, the LDAP host and port information can be discovered from the DNS acting as the source. Here DSD is used to discover host name information from the DNS.

See Also: For detailed reference information and class descriptions, refer to the Javadoc located on the product CD.

SASL Authentication

Oracle Internet Directory supports two mechanisms for SASL-based authentication. This section describes the two methods. It contains these topics:

- SASL Authentication by Using the DIGEST-MD5 Mechanism
- SASL Authentication by Using External Mechanism

SASL Authentication by Using the DIGEST-MD5 Mechanism

SASL Digest-MD5 authentication is the required authentication mechanism for LDAP Version 3 servers (RFC 2829). LDAP Version 2 does not support Digest-MD5.

The Digest-MD5 mechanism is described in RFC 2831 of the Internet Engineering Task Force. It is based on the HTTP Digest Authentication (RFC 2617).

See Also: Internet Engineering Task Force Web site, at <http://www.ietf.org>

This section contains these topics:

- Steps Involved in SASL Authentication by Using DIGEST-MD5
- JAVA APIs for SASL Authentication by Using DIGEST-MD5
- C APIs for SASL authentication using DIGEST-MD5
- SASL Authentication by Using External Mechanism

Steps Involved in SASL Authentication by Using DIGEST-MD5

SASL Digest-MD5 authenticates a user as follows:

1. The directory server sends data that includes various authentication options that it supports and a special token to the LDAP client.
2. The client responds by sending an encrypted response that indicates the authentication options that it has selected. The response is encrypted in such a way that proves that the client knows its password.
3. The directory server then decrypts and verifies the client's response.

To use the Digest-MD5 authentication mechanism, you can use either the Java API or the C API to set up the authentication.

SASL Authentication by Using External Mechanism

The following is from section 7.4 of RFC 2222 of the Internet Engineering Task Force.

The mechanism name associated with external authentication is "EXTERNAL". The client sends an initial response with the authorization identity. The server uses information, external to SASL, to determine whether the client is authorized to authenticate as the authorization identity. If the client is so authorized, the server indicates successful completion of the authentication exchange; otherwise the server indicates failure.

The system providing this external information may be, for example, IPsec or SSL/TLS.

If the client sends the empty string as the authorization identity (thus requesting the authorization identity be derived from the client's authentication credentials), the authorization identity is to be derived from authentication credentials that exist in the system which is providing the external authentication.

Oracle Internet Directory provides the SASL external mechanism over an SSL mutual connection. The authorization identity (DN) is derived from the client certificate during the SSL network negotiation.

Proxying on Behalf of End Users

Often applications must perform operations that require impersonating an end user. An application may, for example, want to retrieve resource access descriptors for an end user. (Resource access descriptors are discussed in the concepts chapter of *Oracle Internet Directory Administrator's Guide*.)

A proxy switch occurs at run time on the JNDI context. An LDAP v3 feature, proxying can only be performed using `InitialLdapContext`, a subclass of `InitialDirContext`. If you use the Oracle extension `oracle.ldap.util.jndi.ConnectionUtil` to establish a connection (the example following), `InitialLdapContext` is always returned. If you use JNDI to establish the connection, make sure that it returns `InitialLdapContext`.

To perform the proxy switch to an end user, the user DN must be available. To learn how to obtain the DN, see the sample implementation of the `oracle.ldap.util.User` class at this URL:

http://www.oracle.com/technology/sample_code/

Look for the Oracle Identity Management link under Sample Applications–Oracle Application Server.

This code shows how the proxy switch occurs:

```
import oracle.ldap.util.jndi.*;
import javax.naming.directory.*;
import javax.naming.ldap.*;
import javax.naming.*;

public static void main(String args[])
{
    try{
        InitialLdapContext appCtx=ConnectionUtil.getDefaultDirCtx(args[0], // host
                                                                    args[1], // port
                                                                    args[2], // DN
                                                                    args[3]; // pass)

        // Do work as application
        // . . .
        String userDN=null;
        // assuming userDN has the end user DN value
        // Now switch to end user
        ctx.addToEnvironment(Context.SECURITY_PRINCIPAL, userDN);
        ctx.addToEnvironment("java.naming.security.credentials", "");
        Control ctls[] = {
            new ProxyControl()
        };
        ((LdapContext)ctx).reconnect(ctls);
        // Do work on behalf of end user
        // . . .
    }
    catch(NamingException ne)
    {
        // javax.naming.NamingException is thrown when an error occurs
    }
}
```

The `ProxyControl` class in the code immediately preceding implements a `javax.naming.ldap.Control`. To learn more about LDAP controls, see the LDAP control section of *Oracle Identity Management User Reference*. Here is an example of what the `ProxyControl` class might look like:

```
import javax.naming.*;
import javax.naming.ldap.Control;
import java.lang.*;

public class ProxyControl implements Control {

    public byte[] getEncodedValue() {
        return null;
    }

    public String getID() {
        return "2.16.840.1.113894.1.8.1";
    }
}
```

```
public boolean isCritical() {  
    return false;  
}  
}
```

Creating Dynamic Password Verifiers

You can modify the LDAP authentication APIs to generate application passwords dynamically—that is, when users log in to an application. This feature has been designed to meet the needs of applications that provide parameters for password verifiers only at runtime.

This section contains the following topics:

- [Request Control for Dynamic Password Verifiers](#)
- [Syntax for DynamicVerifierRequestControl](#)
- [Parameters Required by the Hashing Algorithms](#)
- [Configuring the Authentication APIs](#)
- [Response Control for Dynamic Password Verifiers](#)
- [Obtaining Privileges for the Dynamic Verifier Framework](#)

Request Control for Dynamic Password Verifiers

Creating a password verifier dynamically involves modifying the LDAP authentication APIs `ldap_search` or `ldap_modify` to include parameters for password verifiers. An LDAP control called `DynamicVerifierRequestControl` is the mechanism for transmitting these parameters. It takes the place of the password verifier profile used to create password verifiers statically. Nevertheless, dynamic verifiers, like static verifiers, require that the directory attributes `orclrepasswd` (synchronized case) and `orclunsyncpasswd` (unsynchronized case) be present and that these attributes be populated.

Note that the `orclpwdencryptionenable` attribute of the password policy entry in the user's realm must be set to 1 if `orclrepasswd` is to be generated. If you fail to set this attribute, an exception is thrown when the user tries to authenticate. To generate `orclunsyncpasswd`, you must add the crypto type 3DES to the entry `cn=defaultSharedPINProfileEntry,cn=common,cn=products,cn=oraclecontext`.

Syntax for DynamicVerifierRequestControl

The request control looks like this:

```
DynamicVerifierRequestControl  
controlOid: 2.16.840.1.113894.1.8.14  
criticality: FALSE  
controlValue: an OCTET STRING whose value is the BER encoding of the following  
type:
```

```
ControlValue ::= SEQUENCE {  
  
    version [0]  
    crypto [1] CHOICE OPTIONAL {  
        SASL/MD5 [0] LDAPString,  
        SyncML1.0 [1] LDAPString,  
    }
```

```

        SyncML1.1 [2] LDAPString,
        CRAM-MD5 [3] LDAPString },
    username [1] OPTIONAL LDAPString,
    realm [2] OPTIONAL LDAPString,
    nonce [3] OPTIONAL LDAPString,
}

```

Note that the parameters in the control structure must be passed in the order in which they appear. [Table 3–2](#) defines these parameters.

Table 3–2 Parameters in DynamicVerifierRequestControl

Parameter	Description
controlOID	The string that uniquely identifies the control structure.
crypto	The hashing algorithm. Choose one of the four identified in the control structure.
username	The distinguished name (DN) of the user. This value must always be included.
realm	A randomly chosen realm. It may be the identity management realm that the user belongs to. It may even be an application realm. Required only by the SASL/MD5 algorithm.
nonce	An arbitrary, randomly chosen value. Required by SYNCML1.0 and SYNCML1.1.

Parameters Required by the Hashing Algorithms

[Table 3–3](#) lists the four hashing algorithms that are used to create dynamic password verifiers. The table also lists the parameters that each algorithm uses as building blocks. Note that, although all algorithms use the user name and password parameters, they differ in their use of the `realm` and `nonce` parameters.

Table 3–3 Parameters Required by the Hashing Algorithms

Algorithm	Parameters Required
SASL/MD5	username, realm, password
SYNCML1.0	username, password, nonce
SYNCML1.1	username, password, nonce
CRAM-MD5	username, password

Configuring the Authentication APIs

Applications that require password verifiers to be generated dynamically must include `DynamicVerifierRequestControl` in their authentication APIs. Either `ldap_search` or `ldap_compare` must incorporate the `controlOID` and the control values as parameters. They must BER-encode the control values as shown in "[Syntax for DynamicVerifierRequestControl](#)"; then they must send both `controlOID` and the control values to the directory server.

Parameters Passed If `ldap_search` Is Used

If you want the application to authenticate the user, use `ldap_search` to pass the control structure. If `ldap_search` is used, the directory passes the password verifier that it creates to the client.

`ldap_search` must include the DN of the user, the `controlOID`, and the control values. If the user's password is a single sign-on password, the attribute passed is `authpassword`. If the password is a numeric pin or another type of unsynchronized password, the attribute passed is `orclpasswordverifier;orclcommonpin`.

Parameters Passed If `ldap_compare` Is Used

If you want Oracle Internet Directory to authenticate the user, use `ldap_compare` to pass the control structure. In this case, the directory retains the verifier and authenticates the user itself.

Like `ldap_search`, `ldap_compare` must include the DN of the user, the `controlOID`, the control values, and the user's password attribute. For `ldap_compare`, the password attribute is `orclpasswordverifier;orclcommonpin` (unsynchronized case).

Response Control for Dynamic Password Verifiers

When it encounters an error, the directory sends the LDAP control `DynamicVerifierResponseControl` to the client. This response control contains the error code. To learn about the error codes that the response control sends, see the troubleshooting chapter in *Oracle Internet Directory Administrator's Guide*.

Obtaining Privileges for the Dynamic Verifier Framework

If you want the directory to create password verifiers dynamically, you must add your application identity to the `VerifierServices` group of directory administrators. If you fail to perform this task, the directory returns an `LDAP_INSUFFICIENT_ACCESS` error.

Using the Java API Extensions to JNDI

This chapter explains how to use Java extensions to the standard directory APIs to perform many of the operations introduced in Chapter 3. The chapter presents use cases. The Oracle extensions to the standard APIs are documented in full in *Oracle Internet Directory API Reference*.

The chapter contains the following topics:

- [Sample Code](#)
- [Installing the Java Extensions](#)
- [Using the oracle.java.util Package to Model LDAP Objects](#)
- [The Classes PropertySetCollection, PropertySet, and Property](#)
- [Managing Users](#)
- [Authenticating Users](#)
- [Creating Users](#)
- [Retrieving User Objects](#)
- [Retrieving Objects from Realms](#)
- [Discovering a Directory Server](#)
- [Examples: Java API for Discovering a Directory Server](#)
- [Using DIGEST-MD5 to Perform SASL Authentication](#)

Sample Code

Sample code is available at this URL:

http://www.oracle.com/technology/sample_code/

Look for the Oracle Identity Management link under Sample Applications–Oracle Application Server.

Installing the Java Extensions

The Java extensions are installed along with the standard Java APIs when the LDAP client is installed. The APIs and their extensions are found at `$ORACLE_HOME/jlib/ldapjclnt10.jar`.

Using the oracle.java.util Package to Model LDAP Objects

In Java, LDAP entities—users, groups, realms, and applications—are modeled as Java objects instead of as handles. This modeling is done in the `oracle.java.util` package. All other utility functionality is modeled either as individual objects—as, for example, `GUID`—or as static member functions of a utility class.

For example, to authenticate a user, an application must follow these steps:

1. Create `oracle.ldap.util.user` object, given the user DN.
2. Create a `DirContext` JNDI object with all of the required properties, or get one from a pool of `DirContext` objects.
3. Invoke the `User.authenticate` function, passing in a reference to the `DirContext` object and the user credentials.
4. If the `DirContext` object was retrieved from a pool of existing `DirContext` objects, return it to that pool.

Unlike their C and PL/SQL counterparts, Java programmers do not have to explicitly free objects. The Java garbage collection mechanism performs this task.

The Classes `PropertySetCollection`, `PropertySet`, and `Property`

Many of the methods in the `user`, `subscriber`, and `group` classes return a `PropertySetCollection` object. The object represents a collection of one or more LDAP entries. Each of these entries is represented by a `PropertySet` object, identified by a DN. A property set can contain attributes, each represented as a property. A property is a collection of one or more values for the particular attribute it represents. An example of the use of these classes follows:

```
PropertySetCollection psc = Util.getGroupMembership( ctx,
                                                    myuser,
                                                    null,
                                                    true );

    // for loop to go through each PropertySet
    for (int i = 0; i < psc.size(); i++) {

        PropertySet ps = psc.getPropertySet(i);

        // Print the DN of each PropertySet
        System.out.println("dn: " + ps .getDN());

        // Get the values for the "objectclass" Property
        Property objectclass = ps.getProperty( "objectclass" );

        // for loop to go through each value of Property "objectclass"
        for (int j = 0; j < objectclass.size(); j++) {

            // Print each "objectclass" value
            System.out.println("objectclass: " + objectclass.getValue(j));
        }
    }
}
```

The entity `myuser` is a user object. The `psc` object contains all the nested groups that `myuser` belongs to. The code loops through the resulting entries and prints out all the object class values of each entry.

Managing Users

All user-related functionality is abstracted in a Java class called `oracle.ldap.util.User`. The process works like this:

1. Construct a `oracle.ldap.util.User` object based on a DN, GUID, or simple name.
2. Invoke `User.authenticate(DirContext, Credentials)` to authenticate the user if necessary.
3. Invoke `User.getProperties(DirContext)` to get the attributes of the user entry.
4. Invoke `User.getExtendedProperties(DirContext, PropCategory, PropType)` to get the extended properties of the user. `PropCategory` is either shared or application-specific. `PropType` is the object that represents the type of property desired. If `PropType` is null, all properties in a given category are retrieved.
5. Invoke `PropertyType.getDefinition(DirContext)` to get the metadata required to parse the properties returned in step 4.
6. Parse the extended properties and continue with application-specific logic. This parsing is also performed by application-specific logic.

Authenticating Users

User authentication is a common LDAP operation that compares the credentials that a user provides at login with the user's credentials in the directory. Oracle Internet Directory supports the following:

- Arbitrary attributes can be used during authentication
- Appropriate password policy exceptions are returned by the authentication method. Note, however, that the password policy applies only to the `userpassword` attribute.

The following code fragment shows how the API is used to authenticate a user:

```
// User user1 - is a valid User Object
try
{
    user1.authenticateUser(ctx,
        User.CREDTYPE_PASSWD, "welcome");

    // or
    // user1.authenticateUser(ctx, <any
attribute>, <attribute value>);
}
catch (UtilException ue)
{
    // Handle the password policy error
    accordingly
    if (ue instanceof PasswordExpiredException)
        // do something
    else if (ue instanceof GraceLoginException)
        // do something
}
```

Creating Users

The subscriber class uses the `createUser()` method to programmatically create users. The object classes required by a user entry are configurable through Oracle Delegated Administration Services. The `createUser()` method assumes that the client understands the requirement and supplies the values for the mandatory attributes during user creation. If the programmer does not supply the required information the server will return an error.

The following snippet of sample code demonstrates the usage.

```
// Subscriber sub is a valid Subscriber object
// DirContext ctx is a valid DirContext

// Create ModPropertySet object to define all the attributes and their values.
ModPropertySet mps = new ModPropertySet();
mps.addProperty(LDIF.ATTRIBUTE_CHANGE_TYPE_ADD, "cn", "Anika");
mps.addProperty(LDIF.ATTRIBUTE_CHANGE_TYPE_ADD, "sn", "Anika");
mps.addProperty(LDIF.ATTRIBUTE_CHANGE_TYPE_ADD, "mail",
"Anika@oracle.com");

// Create user by specifying the nickname and the ModPropertySet just defined
User newUser = sub.createUser( ctx, mps);

// Print the newly created user DN
System.out.println( newUser.getDN(ctx) );

// Perform other operations with this new user
```

Retrieving User Objects

The subscriber class offers the `getUser()` method to replace the public constructors of the `User` class. A user object is returned based on the specified information.

The following is a piece of sample code demonstrating the usage:

```
// DirContext ctx is contains a valid directory connection with
sufficient privilege to perform the operations

// Creating RootOracleContext object
RootOracleContext roc = new RootOracleContext(ctx);

// Obtain a Subscriber object representing the default
subscriber
Subscriber sub = roc.getSubscriber(ctx,
Util.IDTYPE_DEFAULT, null, null);

// Obtain a User object representing the user whose
nickname is "Anika"
User user1 = sub.getUser(ctx, Util.IDTYPE_SIMPLE, "Anika",
null);
// Do work with this user
```

The `getUser()` method can retrieve users based on DN, GUID and simple name. A `getUsers()` method is also available to perform a filtered search to return more than one user at a time. The returned object is an array of `User` objects. For example,

```
// Obtain an array of User object where the user's nickname
starts with "Ani"
```



```
User[] userArr = sub.getUsers(ctx, Util.IDTYPE_SIMPLE,
"Ani", null);
// Do work with the User array
```

Retrieving Objects from Realms

This section describes how the Java API can be used to retrieve objects in identity management realms.

The `RootOracleContext` class represents the root Oracle Context. Much of the information needed for identity management realm creation is stored within the root Oracle Context. The `RootOracleContext` class offers the `getSubscriber()` method. It replaces the public constructors of the subscriber class and returns an identity management realm object based on the specified information.

The following is a piece of sample code demonstrating the usage:

```
// DirContext ctx contains a valid directory
// connection with sufficient privilege to perform the
// operations

// Creating RootOracleContext object
RootOracleContext roc = new RootOracleContext(ctx);

// Obtain a Subscriber object representing the
// Subscriber with simple name "Oracle"
Subscriber sub = roc.getSubscriber(ctx,
Util.IDTYPE_SIMPLE, "Oracle", null);

// Do work with the Subscriber object
```

Discovering a Directory Server

A new Java class, the public class, has been introduced:

```
public class oracle.ldap.util.discovery.DiscoveryHelper
```

This class provides a method for discovering specific information from the specified source.

Table 4-1 *Methods for Directory Server Discovery*

Method	Description
<code>discover</code>	Discovers the specific information from a given source
<code>setProperty</code>	Sets the properties required for discovery
<code>getProperty</code>	Accesses the value of properties

Two new methods are added to the existing Java class

```
oracle.ldap.util.jndi.ConnectionUtil:
```

- `getDefaultDirCtx`: This overloaded function determines the host name and port information of non-SSL ldap servers by making an internal call to `oracle.ldap.util.discovery.DiscoveryHelper.discover()`.
- `getSSLDirCtx`: This overloaded function determines the host name and port information of SSL ldap servers by making an internal call to `oracle.ldap.util.discovery.DiscoveryHelper.discover()`.

Examples: Java API for Discovering a Directory Server

The following is a sample Java program for directory server discovery:

```
import java.util.*;
import java.lang.*;
import oracle.ldap.util.discovery.*;
import oracle.ldap.util.jndi.*;

public class dsdtest
{
    public static void main(String s[]) throws Exception
    {
        HashMap reshdl = new HashMap();
        String result = new String();
        Object resultObj = new Object();
        DiscoveryHelper disco = new
DiscoveryHelper(DiscoveryHelper.DNS_DISCOVER);

        // Set the property for the DNS_DN
disco.setProperty(DiscoveryHelper.DNS_DN, "dc=us,dc=fiction,dc=com")
;

        // Set the property for the DNS_DISCOVER_METHOD
disco.setProperty(DiscoveryHelper.DNS_DISCOVER_METHOD,
DiscoveryHelper.USE_INPUT_DN_METHOD);

        // Set the property for the SSLMODE
disco.setProperty(DiscoveryHelper.SSLMODE, "0");

        // Call the discover method
int res=disco.discover(reshdl);
if (res!=0)
    System.out.println("Error Code returned by the discover method is :"+res) ;

        // Print the results
printReshdl(reshdl);
    }

    public static void printReshdl(HashMap reshdl)
    {
        ArrayList result = (ArrayList)reshdl.get(DiscoveryHelper.DIR_SERVERS);

        if (result != null)
        {
            if (result.size() == 0) return;
            System.out.println("The hostnames are :-");
            for (int i = 0; i < result.size();i++)
            {
                String host = (String)result.get(i);
                System.out.println((i+1)+" .
"+"+host+"");
            }
        }
    }
}
```

Using DIGEST-MD5 to Perform SASL Authentication

When using JNDI to create a SASL connection, you must set these `javax.naming.Context` properties:

- `Context.SECURITY_AUTHENTICATION = "DIGEST-MD5"`
- `Context.SECURITY_PRINCIPAL`

The latter sets the principal name. This name is a server-specific format. It can be either of the following:

- The DN—that is, `dn:`—followed by the fully qualified DN of the entity being authenticated
- The string `u:` followed by the user identifier.

The Oracle directory server accepts just a fully qualified DN such as `cn=user,ou=my department,o=my company`.

Note: The SASL DN must be normalized before it is passed to the API that calls the SASL bind. To generate SASL verifiers, Oracle Internet Directory supports only normalized DNs.

Using the API Extensions in PL/SQL

This chapter explains how to use PL/SQL extensions to the standard directory APIs to manage and authenticate users. Note that the Oracle extensions do not include PL/SQL APIs that create users. The Oracle extensions to the standard APIs are documented in full in [Chapter 14](#).

This chapter contains these topics:

- [Sample Code](#)
- [Installing the PL/SQL Extensions](#)
- [Using Handles to Access Directory Data](#)
- [Managing Users](#)
- [Authenticating Users](#)
- [Dependencies and Limitations of the PL/SQL LDAP API](#)

Sample Code

Sample code is available at this URL:

http://www.oracle.com/technology/sample_code/

Look for the Oracle Identity Management link under Sample Applications–Oracle Application Server.

Installing the PL/SQL Extensions

The PL/SQL extensions are installed with the `DBMS_LDAP` package when the Oracle database is installed. You must run the script `$ORACLE_HOME/rdbms/admin/catldap.sql`.

Using Handles to Access Directory Data

Most of the extensions described in this chapter are helper functions. They access data about specific LDAP entities such as users, groups, realms, and applications. In many cases, these functions must pass a reference to one of these entities to the standard API functions. To do this, the API extensions use opaque data structures called handles. The steps that follow show an extension creating a user handle:

1. Establish an LDAP connection or get one from a pool of connections.
2. Create a user handle from user input. This could be a DN, a GUID, or a single sign-on user ID.

3. Authenticate the user with the LDAP connection handle, user handle, or credentials.
4. Free the user handle.
5. Close the LDAP connection, or return the connection back to the connection pool.

Managing Users

The steps that follow show how the `DBMS_LDAP_UTL` package is used to create and use a handle that retrieves user properties from the directory.

1. Invoke `DBMS_LDAP_UTL.create_user_handle(user_hd, user_type, user_id)` to create a user handle from user input. The input can be a DN, a GUID, or a single sign-on user ID.
2. Invoke `DBMS_LDAP_UTL.set_user_handle_properties(user_hd, property_type, property)` to associate a realm with the user handle.
3. Invoke `DBMS_LDAP_UTL.get_user_properties(ld, user_handle, attrs, ptype, ret_pset_coll)` to place the attributes of a user entry into a result handle.
4. Invoke `DBMS_LDAP_UTL.get_property_names(pset, property_names)` and `DBMS_LDAP_UTL.get_property_values(pset, property_name, property_values)` to extract user attributes from the result handle that you obtained in step 3.

Authenticating Users

Use `DBMS_LDAP_UTL.authenticate_user(session, user_handle, auth_type, cred, binary_cred)` to authenticate a user to the directory. This function compares the password provided by the user with the password attribute in the user's directory entry.

Dependencies and Limitations of the PL/SQL LDAP API

The PL/SQL LDAP API for this release has the following limitations:

- The LDAP session handles obtained from the API are valid only for the duration of the database session. The LDAP session handles cannot be written to a table and reused in other database sessions.
- Only synchronous versions of LDAP API functions are supported in this release.

The PL/SQL LDAP API requires a database connection to work. It cannot be used in client-side PL/SQL engines (like Oracle Forms) without a valid database connection.

Developing Provisioning-Integrated Applications

As of 10g Release 2 (10.1.2), new APIs are available for developing provisioning-integrated applications. Please refer to:

- The Oracle Provisioning Service Concepts chapter in *Oracle Identity Management Integration Guide*
- The Deploying Provisioning-Integrated Applications chapter in *Oracle Identity Management Integration Guide*

Developing Directory Plug-ins

This chapter explains how to use the plug-in framework for Oracle Internet Directory to extend LDAP operations.

This chapter contains these topics:

- [Plug-in Prerequisites](#)
- [Plug-in Benefits](#)
- [What Is the Plug-in Framework?](#)
- [Operation-Based Plug-ins Supported by the Directory](#)
- [Designing, Creating, and Using Plug-ins](#)
- [Examples of Plug-ins](#)
- [Binary Support in the Plug-in Framework](#)
- [Database Object Types Defined](#)
- [Specifications for Plug-in Procedures](#)

Plug-in Prerequisites

To develop Oracle Internet Directory plug-ins, you should be familiar with the following:

- Generic LDAP concepts
- Oracle Internet Directory
- Oracle Internet Directory integration with Oracle Application Server
- SQL, PL/SQL, and database RPCs

Plug-in Benefits

To extend the capabilities of the Oracle Internet Directory server, you can write your own server plug-in. A server plug-in is a PL/SQL package, shared object or library, or a dynamic link library on Windows that contains your own functions. Oracle supports only PL/SQL plug-ins.

You can extend LDAP operations in the following ways:

- You can validate data before the server performs an LDAP operation on the data
- You can perform actions (that you define) after the server successfully completes an LDAP operation

- You can define extended operations
- You can be authenticated through external credential stores
- You can replace an existing server module by defining your own server module

For the last one, you may, for example, implement your own password value checking and place it into the Oracle Internet Directory server.

On startup, the directory server loads your plug-in configuration and library. It calls your plug-in functions while processing various LDAP requests.

See Also: The chapter about the password policy plug-in in *Oracle Internet Directory Administrator's Guide*. The chapter contains an example of how to implement your own password value checking and place it into the Oracle Internet Directory server.

What Is the Plug-in Framework?

The plug-in framework is the environment in which the plug-in user can develop, configure, and apply the plug-ins. Each individual plug-in instance is called a plug-in module.

The plug-in framework includes the following:

- Plug-in configuration tools
- Plug-in module interface
- Plug-in LDAP API (ODS.LDAP_PLUGIN package)

Follow these steps to use the server plug-in framework:

1. Write a user-defined plug-in procedure. This plug-in module must be written in PL/SQL.
2. Compile the plug-in module against the same database that serves as the Oracle Internet Directory backend database.
3. Register the plug-in module through the configuration entry interface.

Operation-Based Plug-ins Supported by the Directory

For operation-based plug-ins, there are pre-operation, post-operation, and when-operation plug-ins.

Pre-Operation Plug-ins

The server calls pre-operation plug-in modules before performing the LDAP operation. The main purpose of this type of plug-in is to validate data before the data can be used in the LDAP operation.

When an exception occurs in the pre-operation plug-in, one of the following occurs:

- When the return error code indicates warning status, the associated LDAP request proceeds.
- When the return code indicates failure status, the request does not proceed.

If the associated LDAP request fails later on, the directory does not roll back the committed code in the plug-in modules.

Post-Operation Plug-ins

The Oracle Internet Directory server calls post-operation plug-in modules after performing an LDAP operation. The main purpose of this type of plug-in is to invoke a function after a particular LDAP operation is executed. For example, logging and notification are post-operation plug-in functions.

When an exception occurs in the post-operation plug-in, the associated LDAP operation is not rolled back.

If the associated LDAP request fails, the post plug-in is still executed.

When-Operation Plug-ins

The directory calls when-operation plug-in modules while performing standard LDAP operations. The main purpose of this type of plug-in is to augment existing operations within the same LDAP transaction. If either the LDAP request or the plug-in program fails, all the changes are rolled back.

There are different types of When-operation plug-ins.

- Add-on
- Replace

You can, for example, use both add-on and replace plug-ins with the `ldapcompare` operation. If you use the first type, the directory executes its server compare code and executes the plug-in module defined by the plug-in developer. If you use the second type, the directory does not execute its compare code. Instead it relies on the plug-in module to perform the comparison.

Replace plug-ins are supported only in `ldapadd`, `ldapcompare`, `ldapdelete`, `ldapmodify`, and `ldapbind`. Add-on plug-ins are supported in `ldapadd`, `ldapdelete`, and `ldapmodify`.

Designing, Creating, and Using Plug-ins

This section contains these topics:

- [Designing Plug-ins](#)
- [Creating Plug-ins](#)
- [Compiling Plug-ins](#)
- [Registering Plug-ins](#)
- [Managing Plug-ins](#)
- [Enabling and Disabling Plug-ins](#)
- [Exception Handling](#)
- [Plug-in LDAP API](#)
- [Plug-ins and Replication](#)
- [Plug-in and Database Tools](#)
- [Security](#)
- [Plug-in Debugging](#)
- [Plug-in LDAP API Specifications](#)
- [Database Limitations](#)

Designing Plug-ins

Use the following guidelines when designing plug-ins:

- Use plug-ins to guarantee that when a specific LDAP operation is performed, related actions are also performed.
- Use plug-ins only for centralized, global operations that should be invoked for the program body statement, regardless of which user or LDAP application issues the statement.
- Do not create recursive plug-ins. For example, creating a `PRE_LDAP_BIND` plug-in that itself issues an `ldapbind` (through the `DBMS_LDAP` PL/SQL API) statement, causes the plug-in to execute recursively until it has run out of resources.

Note: Use plug-ins on the LDAP PL/SQL API judiciously. They are executed for every LDAP request every time the event occurs on which the plug-in is created.

Types of Plug-in Operations

A plug-in can be associated with `ldapbind`, `ldapadd`, `ldapmodify`, `ldapcompare`, `ldapsearch`, and `ldapdelete` operations.

Naming Plug-ins

Plug-in names (PL/SQL package names) must be unique if they share the same database schema with other plug-ins or stored procedures. But plug-ins can share names with other database schema objects such as tables and views. This kind of sharing is not, however, recommended.

Creating Plug-ins

Creating a plug-in module is like creating a PL/SQL package. Both have a specification part and a body part. The directory, not the plug-in, defines the plug-in specification because the specification serves as the interface between Oracle Internet Directory and the custom plug-in.

For security reasons and for the integrity of the LDAP server, you can compile plug-ins only in the ODS database schema. You must compile them in the database that serves as the backend database of Oracle Internet Directory.

Package Specifications for Plug-in Module Interfaces

Different plug-ins have different package specifications. As [Table 7-1](#) shows, you can name the plug-in package. You must, however, follow the signatures defined for each type of plug-in procedure. See "[Specifications for Plug-in Procedures](#)" for details.

Table 7-1 Plug-in Module Interface

Plug-in Item	User Defined	Oracle Internet Directory-Defined
Plug-in Package Name	X	
Plug-in Procedure Name		X
Plug-in Procedure Signature		X

Table 7-2 names the different plug-in procedures. In addition, it lists and describes the parameters that these procedures use.

Table 7-2 Operation-Based and Attribute-Based Plug-in Procedure Signatures

Invocation Context	Procedure Name	IN Parameters	OUT Parameters
Before ldapbind	PRE_BIND	ldapcontext, Bind DN, Password	return code, error message
With ldapbind but replacing the default server behavior	WHEN_BIND_REPLACE	ldapcontext, bind result, DN, userpassword	bind result, return code, error message
After ldapbind	POST_BIND	ldapcontext, Bind result, Bind DN, Password	return code, error message
Before ldapmodify	PRE_MODIFY	ldapcontext, DN, Mod structure	return code, error message
With ldapmodify	WHEN_MODIFY	ldapcontext, DN, Mod structure	return code, error message
With ldapmodify but replacing the default server behavior	WHEN_MODIFY_REPLACE	ldapcontext, DN, Mod structure	return code, error message
After ldapmodify	POST_MODIFY	ldapcontext, Modify result, DN, Mod structure	return code, error message
Before ldapcompare	PRE_COMPARE	ldapcontext, DN, attribute, value	return code, error message
With ldapcompare but replacing the default server behavior	WHEN_COMPARE_REPLACE	ldapcontext, Compare result, DN, attribute, value	compare result, return code, error message
After ldapcompare	POST_COMPARE	ldapcontext, Compare result, DN, attribute, value	return code, error message
Before ldapadd	PRE_ADD	ldapcontext, DN, Entry	return code, error message
With ldapadd	WHEN_ADD	ldapcontext, DN, Entry	return code, error message
With ldapadd but replacing the default server behavior	WHEN_ADD_REPLACE	ldapcontext, DN, Entry	return code, error message
After ldapadd	POST_ADD	ldapcontext, Add result, DN, Entry	return code, error message
Before ldapdelete	PRE_DELETE	ldapcontext, DN	return code, error message
With ldapdelete	WHEN_DELETE	ldapcontext, DN	return code, error message
With ldapdelete but replacing the default server behavior	WHEN_DELETE	ldapcontext, DN	return code, error message
After ldapdelete	POST_DELETE	ldapcontext, Delete result, DN	return code, error message
Before ldapsearch	PRE_SEARCH	ldapcontext, Base DN, scope, filter	return code, error message

Table 7–2 (Cont.) Operation-Based and Attribute-Based Plug-in Procedure Signatures

Invocation Context	Procedure Name	IN Parameters	OUT Parameters
After ldapsearch	POST_SEARCH	Ldap context, Search result, Base DN, scope, filter	return code, error message

See Also:

- ["Error Handling"](#) on page 7-10 for valid values for the return code and error message
- ["Specifications for Plug-in Procedures"](#) on page 7-26 for complete supported procedure signatures

Compiling Plug-ins

Plug-ins are exactly the same as PL/SQL stored procedures. A PL/SQL anonymous block is compiled each time it is loaded into memory. Compilation consists of these stages:

1. Syntax checking: PL/SQL syntax is checked, and a parse tree is generated.
2. Semantic checking: Type checking and further processing on the parse tree.
3. Code generation: The pcode is generated.

If errors occur during the compilation of a plug-in, the plug-in is not created. You can use the `SHOW ERRORS` statement in SQL*Plus or Enterprise Manager to see any compilation errors when you create a plug-in, or you can `SELECT` the errors from the `USER_ERRORS` view.

All plug-in modules must be compiled in the ODS database schema.

Dependencies

Compiled plug-ins have dependencies. They become invalid if an object depended upon, such as a stored procedure or function called from the plug-in body, is modified. Plug-ins that are invalidated for dependency reasons must be recompiled before the next invocation.

Recompiling Plug-ins

Use the `ALTER PACKAGE` statement to manually recompile a plug-in. For example, the following statement recompiles the `my_plugin` plug-in:

```
ALTER PACKAGE my_plugin COMPILE PACKAGE;
```

Registering Plug-ins

To enable the directory server to call a plug-in at the right moment, you must register the plug-in with the directory server. Do this by creating an entry for the plug-in under `cn=plugin,cn=subconfigsubentry`.

The `orclPluginConfig` Object Class

A plug-in must have `orclPluginConfig` as one of its object classes. This is a structural object class, and its super class is `top`. [Table 7–3](#) lists and describes its attributes.

Table 7-3 Plug-in Attribute Names and Values

Attribute Name	Attribute Value	Mandatory?
cn	Plug-in entry name	Yes
orclPluginAttributeList	A semicolon-separated list of attribute names that controls whether the plug-in takes effect. If the target attribute is included in the list, then the plug-in is invoked. Only for ldapcompare and ldapmodify plug-ins.	No
orclPluginEnable	0 = disable (default) 1 = enable	No
orclPluginEntryProperties	An ldapsearch filter type value must be specified. For example, if we specify <code>orclPluginEntryProperties: (&(objectclass=inetorgperson)(sn=Cezanne))</code> , the plug-in is not invoked if the target entry has <code>objectclass</code> equal to <code>inetorgperson</code> and <code>sn</code> equal to <code>Cezanne</code> .	No
orclPluginIsReplace	0 = disable (default) 1 = enable For WHEN timing plug-in only	No
orclPluginKind	PL/SQL	No
orclPluginLDAPOperation	One of the following values: ldapcompare ldapmodify ldapbind ldapadd ldapdelete ldapsearch	Yes
orclPluginName	Plug-in package name	Yes
orclPluginRequestGroup	A semicolon-separated group list that controls if the plug-in takes effect. You can use this group to specify who can actually invoke the plug-in. For example, if you specify <code>orclpluginrequestgroup:cn=security,cn=groups,dc=oracle,dc=com</code> when you register the plug-in, the plug-in will not be invoked unless the ldap request comes from the person who belongs to the group <code>cn=security,cn=groups,dc=oracle,dc=com</code> .	No
orclPluginRequestNegGroup	A semicolon-separated group list that controls if the plug-in takes effect. You can use this group to specify who cannot invoke the plug-in. For example, if you specify <code>orclpluginrequestgroup:cn=security,cn=groups,dc=oracle,dc=com</code> , when you register the plug-in, the plug-in is not invoked if the LDAP request comes from the person who belongs to the group <code>cn=security,cn=groups,dc=oracle,dc=com</code> .	No

Table 7–3 (Cont.) Plug-in Attribute Names and Values

Attribute Name	Attribute Value	Mandatory?
orclPluginResultCode	An integer value to specify the ldap result code. If this value is specified, then plug-in will be invoked only if the LDAP operation is in that result code scenario. This is only for the post plug-in type.	No
orclPluginShareLibLocation	File location of the dynamic linking library. If this value is not present, then Oracle Internet Directory server assumes the plug-in language is PL/SQL.	No
orclPluginSubscriberDNList	A semicolon separated DN list that controls if the plug-in takes effect. If the target DN of an LDAP operation is included in the list, then the plug-in is invoked.	No
orclPluginTiming	One of the following values: pre when post	No
orclPluginType	One of the following values: operational attribute password_policy syntax matchingrule See Also: "Operation-Based Plug-ins Supported by the Directory" on page 7-2	Yes
orclPluginVersion	Supported plug-in version number	No

Adding a Plug-in Configuration Entry by Using Command-Line Tools

Plug-ins must be added to Oracle Internet Directory server so that the server is aware of additional operations that must be performed at the correct time.

When the plug-in successfully compiles against the Oracle Internet Directory backend database, create a new entry and place it under `cn=plugin,cn=subconfigsubentry`.

In the following examples, an entry is created for an operation-based plug-in called `my_plugin1`. The LDIF file, `my_ldif_file.ldif`, is as follows:

Example 1

The following is an example LDIF file to create such an object:

```
cn=when_comp,cn=plugin,cn=subconfigsubentry
objectclass=orclPluginConfig
objectclass=top
orclPluginName=my_plugin1
orclPluginType=operational
orclPluginTiming=when
orclPluginLDAPOperation=ldapcompare
orclPluginEnable=1
orclPluginVersion=1.0.1
orclPluginIsReplace=1
cn=when_comp
orclPluginKind=PLSQL
```



```

orclPluginSubscriberDNList=dc=COM,c=us;dc=us,dc=oracle,dc=com;dc=org,dc=us;o=IMC
,c=US
orclPluginAttributeList=userpassword

```

Example 2

```

cn=post_mod_plugin, cn=plugin,cn=subconfigsentry
objectclass=orclPluginConfig
objectclass=top
orclPluginName=my_plugin1
orclPluginType=operational
orclPluginTiming=post
orclPluginLDAPOperation=ldapmodify
orclPluginEnable=1
orclPluginVersion=1.0.1
cn=post_mod_plugin
orclPluginKind=PLSQL

```

Add this file to the directory with the following command:

```

ldapadd -p 389 -h myhost -D binddn -w password -f my_ldif_file.ldif

```

Note: The plug-in configuration entry is not replicated. Replicating it would create an inconsistent state.

Managing Plug-ins

This section explains how to modify and debug plug-ins.

Modifying Plug-ins

Like a stored procedure, a plug-in cannot be explicitly altered. It must be replaced with a new definition.

When replacing a plug-in, you must include the `OR REPLACE` option in the `CREATE PACKAGE` statement. The `OR REPLACE` option enables a new version of an existing plug-in to replace an older version without having an effect on grants made for the original version of the plug-in.

Alternatively, the plug-in can be dropped using the `DROP PACKAGE` statement, and you can rerun the `CREATE PACKAGE` statement.

If the plug-in name (the package name) is changed, you must register the new plug-in again.

Debugging Plug-ins

You can debug a plug-in using the same facilities available for PL/SQL stored procedures.

Enabling and Disabling Plug-ins

To turn the plug-in on or off, modify the value of `orclPluginEnable` in the plug-in configuration object. For example, modify the value of `orclPluginEnable` in `cn=post_mod_plugin, cn=plugins, cn=subconfigsentry` to be 1 or 0.

Exception Handling

Each of the procedures in a PL/SQL plug-in must have an exception handling block that handles errors intelligently and, if possible, recovers from them.

Error Handling

Oracle Internet Directory requires that the return code (`rc`) and error message (`errmsg`) be set correctly in the plug-in procedures.

[Table 7-4](#) provides the values that are valid for the return code.

Table 7-4 Valid Values for the plug-in Return Code

Error Code	Description
0	Success
Any number greater than zero	Failure
-1	Warning

The `errmsg` parameter is a string value that can pass a user's custom error message back to Oracle Internet Directory server. The size limit for `errmsg` is 1024 bytes. Each time Oracle Internet Directory runs the plug-in program, it examines the return code to determine if it must display the error message.

If, for example, the value for the return code is 0, the error message value is ignored. If the value of the return code is -1 or greater than zero, the following message is either logged in the log file or displayed in standard output if the request came from LDAP command-line tools:

```
ldap addition info: customized error
```

Program Control Handling between Oracle Internet Directory and Plug-ins

[Table 7-5](#) shows where plug-in exceptions occur and how the directory handles them.

Table 7-5 Program Control Handling when a Plug-in Exception Occurs

Plug-in Exception Occurred in	Oracle Internet Directory Server Handling
PRE_BIND, PRE_MODIFY, PRE_ADD, PRE_SEARCH, PRE_COMPARE, PRE_DELETE	Depends on return code. If the return code is: <ul style="list-style-type: none"> ■ Greater than zero (error), then no LDAP operation is performed ■ -1 (warning), then proceed with the LDAP operation
POST_BIND, POST_MODIFY, POST_ADD, POST_SEARCH, WHEN_DELETE	LDAP operation is completed. There is no rollback.
WHEN_MODIFY, WHEN_ADD, WHEN_DELETE	Rollback the LDAP operation

[Table 7-6](#) shows how the directory responds when an LDAP operation fails.

Table 7–6 Program Control Handling when an LDAP Operation Fails

LDAP Operation Fails in	Oracle Internet Directory Server Handling
PRE_BIND, PRE_MODIFY, PRE_ADD, PRE_SEARCH, WHEN_DELETE	Pre-operation plug-in is completed. There is no rollback.
POST_BIND, POST_ MODIFY, POST_ADD, POST_SEARCH, WHEN_ DELETE	Proceed with post-operation plug-in. The LDAP operation result is one of the IN parameters.
WHEN_MODIFY, WHEN_ ADD, WHEN_DELETE	When types of plug-in changes are rolled back.
WHEN	Changes made in the plug-in program body are rolled back.

Plug-in LDAP API

There are different methods for providing API access:

- Enable a user to utilize the standard LDAP PL/SQL APIs. Note though that, if program logic is not carefully planned, an infinite loop in plug-in execution can result.
- Oracle Internet Directory provides the Plug-in LDAP API. This plug-in does not cause a series of plug-in actions in the directory server if there are plug-ins configured and associated with the LDAP request.

In the Plug-in LDAP API, the directory provides APIs for connecting back to the directory server designated in the plug-in module. You must use this API if you want to connect to the server that is executing the plug-in. If you want to connect to an external server, you can use the DBMS_LDAP API.

Within each plug-in module, an `ldapcontext` is passed from the Oracle directory server. When the Plug-in LDAP API is called, `ldapcontext` is passed for security and binding purposes. When binding with this `ldapcontext`, Oracle Internet Directory recognizes that the LDAP request is coming from a plug-in module. For this type of plug-in bind, the directory does not trigger any subsequent plug-ins. It handles the plug-in bind as a super-user bind. Use this plug-in bind with discretion.

See Also: ["Plug-in LDAP API Specifications"](#) on page 7-12

Plug-ins and Replication

These cases can cause an inconsistent state in a replication environment:

- Plug-in metadata replicated to other nodes
- Changes to directory entries by plug-in programs or other LDAP operations
- Plug-in installation on only some of the participating nodes
- Implementation in the plug-in of extra checking that depends on the directory data

Plug-in and Database Tools

Bulk tools do not support server plug-ins.

Security

Some Oracle Internet Directory server plug-ins require that you supply the code that preserves tight security. For example, if you replace the directory's `ldapcompare` or `ldapbind` operation with your own plug-in module, you must ensure that your implementation of this operation does not omit any functionality on which security relies.

To ensure tight security, the following must be done:

- Create the plug-in packages
- Only the LDAP administrator can restrict the database user
- Use the access control list (ACL) to set the plug-in configuration entries to be accessed only by the LDAP administrator
- Be aware of the program relationship between different plug-ins

Plug-in Debugging

Use the plug-in debugging mechanism for Oracle Internet Directory to examine the process and content of plug-ins. The following commands control the operation of the server debugging process.

- To set up plug-in debugging, run this command:

```
% sqlplus ods/password @$ORACLE/ldap/admin/oidspdsu.pls
```

- To enable plug-in debugging, run this command:

```
% sqlplus ods/password @$ORACLE/ldap/admin/oidspdon.pls
```

- After enabling plug-in debugging, you can use this command in the plug-in module code:

```
plg_debug('debuggingmessage');
```

The resulting debug message is stored in the plug-in debugging table.

- To disable debugging, run this command:

```
% sqlplus ods/password @$ORACLE/ldap/admin/oidsp dof.pls
```

- To display the debug messages that you put in the plug-in module, run this command:

```
% sqlplus ods/password @$ORACLE/ldap/admin/oidsp dsh.pls
```

- To delete all of the debug messages from the debug table, run this command:

```
% sqlplus ods/password @$ORACLE/ldap/admin/oidsp dde.pls
```

Plug-in LDAP API Specifications

Here is the package specification that Oracle Internet Directory provides for the Plug-in LDAP API:

```
CREATE OR REPLACE PACKAGE LDAP_PLUGIN AS
  SUBTYPE SESSION IS RAW(32);

  -- Initializes the LDAP library and return a session handler
  -- for use in subsequent calls.
  FUNCTION init (ldappluginctx IN ODS.plugincontext)
```

```

RETURN SESSION;

-- Synchronously authenticates to the directory server using
-- a Distinguished Name and password.
FUNCTION simple_bind_s (ldappluginctx IN ODS.plugincontext,
                       ld              IN SESSION)
RETURN PLS_INTEGER;

-- Get requester info from the plug-in context
FUNCTION get_requester (ldappluginctx IN ODS.plugincontext)
RETURN VARCHAR2;
END LDAP_PLUGIN;
```

Database Limitations

Oracle Internet Directory 10g Release 2 (10.1.2) can use several different versions of the Oracle Database for storing directory data. These include Oracle9i Database Server Release 2, v9.2.0.6 or later and Oracle Database 10g, v10.1.0.4 or later.

In Oracle Application Server 10g Release 2 (10.1.2), the following plug-in features are not supported in the directory server running against Oracle9i Database Server Release 2:

- Windows Domain external authentication plug-in.
- The `simple_bind_s()` function of the `LDAP_PLUGIN` package provided as the Oracle Internet Directory PL/SQL PLUGIN API for connecting back to the directory server as part of plug-in definitions.

Examples of Plug-ins

This section presents two sample plug-ins. One logs all `ldapsearch` commands. The other synchronizes two directory information trees (DITs).

Example 1: Search Query Logging

Situation: A user wants to know if it is possible to log all of the `ldapsearch` commands.

Solution: Yes. The user can use the `post_ldapsearch` operational plug-in for this purpose. They can either log all of the requests or only those that occur under the DNs being searched.

To log all the `ldapsearch` commands:

1. Log all of the `ldapsearch` results into a database table. This log table has these columns:
 - timestamp
 - baseDN
 - search scope
 - search filter
 - required attribute
 - search result

Use this SQL script to create the table:

```
drop table search_log;
```

```
create table search_log
(timestamp varchar2(50),
basedn varchar2(256),
searchscope number(1);
searchfilter varchar2(256);
searchresult number(1));
drop table simple_tab;
create table simple_tab (id NUMBER(7), dump varchar2(256));
DROP sequence seq;
CREATE sequence seq START WITH 10000;
commit;
```

2. Create the plug-in package specification.

```
CREATE OR REPLACE PACKAGE LDAP_PLUGIN_EXAMPLE1 AS
PROCEDURE post_search
(ldapplugincontext IN ODS.plugincontext,
result            IN  INTEGER,
basedN           IN  VARCHAR2,
scope            IN  INTEGER,
filterStr        IN  VARCHAR2,
requiredAttr     IN  ODS.strCollection,
rc               OUT INTEGER,
errormsg         OUT VARCHAR2
);
END LDAP_PLUGIN_EXAMPLE1;
/
```

3. Create the plug-in package body.

```
CREATE OR REPLACE PACKAGE BODY LDAP_PLUGIN_EXAMPLE1 AS
PROCEDURE post_search
(ldapplugincontext IN ODS.plugincontext,
result            IN  INTEGER,
basedN           IN  VARCHAR2,
scope            IN  INTEGER,
filterStr        IN  VARCHAR2,
requiredAttr     IN  ODS.strCollection,
rc               OUT INTEGER,
errormsg         OUT VARCHAR2
)
IS
BEGIN
    INSERT INTO simple_tab VALUES
(to_char(sysdate, 'Month DD, YYYY HH24:MI:SS'), basedN, scope, filterStr,
result);
    -- The following code segment demonstrate how to iterate
    -- the ODS.strCollection
    FOR l_counter1 IN 1..requiredAttr.COUNT LOOP
        INSERT INTO simple_tab
            values (seq.NEXTVAL, 'req attr ' || l_counter1 || ' = ' ||
                requiredAttr(l_counter1));
    END LOOP;
    rc := 0;
    errormsg := 'no post_search plug-in error msg';
    COMMIT;
EXCEPTION
    WHEN others THEN
        rc := 1;
        errormsg := 'exception: post_search plug-in';
END;
```

```
END LDAP_PLUGIN_EXAMPLE1;
/
```

4. Register the plug-in entry in Oracle Internet Directory.

```
cn=post_search,cn=plugin,cn=subconfigsentry
objectclass=orclPluginConfig
objectclass=top
orclPluginName=ldap_plugin_example1
orclPluginType=operational
orclPluginTiming=post
orclPluginLDAPOperation=ldapsearch
orclPluginEnable=1
orclPluginVersion=1.0.1
cn=post_search
orclPluginKind=PLSQL
```

Using the `ldapadd` command-line tool to add this entry:

```
% ldapadd -p port_number -h host_name -D bind_dn -w passwd -v \
-f register_post_search.ldif
```

Example 2: Synchronizing Two DITs

Situation: There are two interdependent products under `cn=Products`, `cn=oraclecontext`. This interdependency extends down to the users in these products' containers. If a user in the first DIT (product 1) is deleted, the corresponding user in the other DIT (product 2) must be deleted.

Is it possible to set a trigger that, when the user in the first DIT is deleted, calls or passes a trigger to delete the user in the second DIT?

Solution: Yes, we can use the `post ldapdelete` operation plug-in to handle the second deletion occurring in the second DIT.

If the first DIT has the naming context of

`cn=DIT1,cn=products,cn=oraclecontext` and the second DIT has the naming context of `cn=DIT2,cn=products,cn=oraclecontext`, the two users share the same ID attribute. Inside of the `post ldapdelete` plug-in module, we can use `LDAP_PLUGIN` and `DBMS_LDAP` APIs to delete the user in the second DIT.

We must set `orclPluginSubscriberDNList` to

`cn=DIT1,cn=products,cn=oraclecontext`, so that whenever we delete entries under `cn=DIT1,cn=products,cn=oraclecontext`, the plug-in module is invoked.

Note: When you use a post `ldapmodify` plug-in to synchronize changes between two Oracle Internet Directory nodes, you cannot push all the attributes from one node to the other. This is because the changes (mod structure) captured in the plug-in module include operational attributes. These operational attributes are generated on each node and cannot be modified by using the standard LDAP methods.

When writing your plug-in program, exclude the following operational attributes from synchronization: `authPassword`, `creatorsname`, `createtimestamp`, `modifiersname`, `modifytimestamp`, `pwdchangedtime`, `pwdfailuretime`, `pwdaccountlockedtime`, `pwdexpirationwarned`, `pwdreset`, `pwdhistory`, `pwdgraceusetime`.

The following attributes are used the most in the deployment environment and should be excluded from synchronization first: `pwdchangedtime`, `pwdfailuretime`, `authpassword`, `pwdaccountlockedtime`.

1. Assume that the entries under both DITs have been added to the directory. For example, the entry `id=12345,cn=DIT1,cn=products,cn=oraclecontext` is in DIT1, and `id=12345,cn=DIT2,cn=products,cn=oraclecontext` is in DIT2.
2. Create the plug-in package specification.

```
CREATE OR REPLACE PACKAGE LDAP_PLUGIN_EXAMPLE2 AS
PROCEDURE post_delete
(ldapplugincontext IN ODS.plugincontext,
result IN INTEGER,
dn IN VARCHAR2,
rc OUT INTEGER,
errormsg OUT VARCHAR2
);
END LDAP_PLUGIN_EXAMPLE2;
/
```

3. Create the plug-in package body.

```
CREATE OR REPLACE PACKAGE BODY LDAP_PLUGIN_EXAMPLE2 AS
PROCEDURE post_delete
(ldapplugincontext IN ODS.plugincontext,
result IN INTEGER,
dn IN VARCHAR2,
rc OUT INTEGER,
errormsg OUT VARCHAR2
)
IS
    retval PLS_INTEGER;
    my_session DBMS_LDAP.session;
    newDN VARCHAR2(256);
BEGIN
    retval := -1;
    my_session := LDAP_PLUGIN.init(ldapplugincontext);
    -- bind to the directory
    retval := LDAP_PLUGIN.simple_bind_s(ldapplugincontext, my_session);
    -- if retval is not 0, then raise exception
    newDN := REPLACE(dn, 'DIT1', 'DIT2');
END;
```



```

        retval := DBMS_LDAP.delete_s(my_session, newDN);
        -- if retval is not 0, then raise exception
        rc := 0;
        errmsg := 'no post_delete plug-in error msg';
EXCEPTION
    WHEN others THEN
        rc := 1;
        errmsg := 'exception: post_delete plug-in';
END;
END LDAP_PLUGIN_EXAMPLE2;
/
(ldapplugincontext IN ODS.plugincontext,
result IN INTEGER,
dn IN VARCHAR2,
rc OUT INTEGER,
errmsg OUT VARCHAR2
)
IS
    retval PLS_INTEGER;
    my_session DBMS_LDAP.session;
    newDN VARCHAR2(256);
BEGIN
    retval := -1;
    my_session := LDAP_PLUGIN.init(ldapplugincontext);
    -- bind to the directory
    retval := LDAP_PLUGIN.simple_bind_s(ldapplugincontext, my_session);
    -- if retval is not 0, then raise exception
    newDN := REPLACE(dn, 'DIT1', 'DIT2');
    retval := DBMS_LDAP.delete_s(my_session, newDN);
    -- if retval is not 0, then raise exception
    rc := 0;
    errmsg := 'no post_delete plug-in error msg';
EXCEPTION
    WHEN others THEN
        rc := 1;
        errmsg := 'exception: post_delete plug-in';
END;
END LDAP_PLUGIN_EXAMPLE2;
/

```

4. Register the plug-in entry with Oracle Internet Directory.

Construct the LDIF file `register_post_delete.ldif`:

```

cn=post_delete,cn=plugin,cn=subconfigsubentry
objectclass=orclPluginConfig
objectclass=top
orclPluginName=ldap_plugin_example2
orclPluginType=operational
orclPluginTiming=post
orclPluginLDAPOperation=ldapdelete
orclPluginEnable=1
orclPluginSubscriberDNList=cn=DIT1,cn=oraclecontext,cn=products
orclPluginVersion=1.0.1
cn=post_delete
orclPluginKind=PLSQL

```

Use the `ldapadd` command-line tool to add this entry:

```

% ldapadd -p port_number -h host_name -D bind_dn -w passwd -v -f register_
post_delete.ldif

```

Binary Support in the Plug-in Framework

Starting with release 10.1.2, object definitions in the Plug-in LDAP API enable `ldapmodify`, `ldapadd`, and `ldapcompare` plug-ins to access binary attributes in the directory database. Formerly, only attributes of type `VARCHAR2` could be accessed. These object definitions do not invalidate plug-in code that precedes release 10.1.2. No change to this code is required. The new definitions appear in the section "[Database Object Types Defined](#)".

The section that you are reading now examines binary operations involving the three types of plug-ins. It includes examples of these plug-ins. The new object definitions apply to pre, post, and when versions of all three.

Note that the three examples use RAW functions and variables in place of LOBs.

Binary Operations with `ldapmodify`

The `modobj` object that the plug-in framework passes to an `ldapmodify` plug-in now holds the values of binary attributes as `binvals`. This variable is a table of `binvalobj` objects.

The plug-in determines whether a binary operation is being performed by examining the `operation` field of `modobj`. It checks whether any of the values `DBMS_LDAP.MOD_ADD`, `DBMS_LDAP.MOD_DELETE`, and `DBMS_LDAP.MOD_REPLACE` are paired with `DBMS_LDAP.MOD_BVALUES`. The pairing `DBMS_LDAP.MOD_ADD+DBMS_LDAP.MOD_BVALUES`, for example, signifies a binary add in the modify operation.

The example that follows shows a post `ldapmodify` plug-in modifying an entry in another directory. The plug-in is invoked after `ldapmodify` applies the same change to the same entry in the plug-in directory. The entry in the other directory appears under the DIT `cn=users,dc=us,dc=acme,dc=com`.

```
create or replace package moduser as
  procedure post_modify(ldapplugincontext IN ODS.plugincontext,
    result IN integer,
    dn IN varchar2,
    mods IN ODS.modlist,
    rc OUT integer,
    errmsg OUT varchar2);
end moduser;
/
show error

CREATE OR REPLACE PACKAGE BODY moduser AS
  procedure post_modify(ldapplugincontext IN ODS.plugincontext,
    result IN integer,
    dn IN varchar2,
    mods IN ODS.modlist,
    rc OUT integer,
    errmsg OUT varchar2)
  is
    counter1 pls_integer;
    counter2 pls_integer;
    retval pls_integer := -1;
    user_session DBMS_LDAP.session;
    user_dn varchar(256);
    user_array DBMS_LDAP.mod_array;
    user_vals DBMS_LDAP.string_collection;
    user_binvals DBMS_LDAP.blob_collection;
    ldap_host varchar(256);
```

```

ldap_port varchar(256);
ldap_user varchar(256);
ldap_passwd varchar(256);
begin
ldap_host := 'backup.us.oracle.com';
ldap_port := '4000';
ldap_user := 'cn=orcladmin';
ldap_passwd := 'welcome';

plg_debug('START MODIFYING THE ENTRY');

-- Get a session
user_session := dbms_ldap.init(ldap_host, ldap_port);

-- Bind to the directory
retval := dbms_ldap.simple_bind_s(user_session, ldap_user,
ldap_passwd);

-- Create a mod_array
user_array := dbms_ldap.create_mod_array(mods.count);

-- Create a user_dn
user_dn := substr(dn,1,instr(dn,',',1,1))||'cn=users,dc=us,dc=acme,
dc=com';

plg_debug('THE CREATED DN IS' || user_dn);

-- Iterate through the modlist
for counter1 in 1..mods.count loop

-- Log the attribute name and operation
if (mods(counter1).operation > DBMS_LDAP.MOD_BVALUES) then
plg_debug('THE NAME OF THE BINARY ATTR. IS' || mods(counter1).type);
else
plg_debug('THE NAME OF THE NORMAL ATTR. IS' || mods(counter1).type);
end if;
plg_debug('THE OPERATION IS' || mods(counter1).operation);

-- Add the attribute values to the collection
for counter2 in 1..mods(counter1).vals.count loop
user_vals(counter2) := mods(counter1).vals(counter2).val;
end loop;

-- Add the attribute values to the collection
for counter2 in 1..mods(counter1).binvals.count loop
plg_debug('THE NO. OF BYTES OF THE BINARY ATTR. VALUE IS'
|| mods(counter1).binvals(counter2).length);
user_binvals(counter2) := mods(counter1).binvals(counter2).binval;
end loop;

-- Populate the mod_array accordingly with binary/normal attributes
if (mods(counter1).operation >= DBMS_LDAP.MOD_BVALUES) then
dbms_ldap.populate_mod_array(user_array,mods(counter1).operation -
DBMS_LDAP.MOD_BVALUES,mods(counter1).type,user_binvals);
user_binvals.delete;
else
dbms_ldap.populate_mod_array(user_array,mods(counter1).operation,
mods(counter1).type,user_vals);
user_vals.delete;
end if;
end if;

```

```
        end loop;

-- Modify the entry
    retval := dbms_ldap.modify_s(user_session,user_dn,user_array);
    if retval = 0 then
        rc := 0;
        errormsg:='No error ocured while modifying the entry';
    else
        rc := retval;
        errormsg :='Error code'||rc||' while modifying the entry';
    end if;

-- Free the mod_array
    dbms_ldap.free_mod_array(user_array);

    plg_debug('FINISHED MODIFYING THE ENTRY');

    exception
    WHEN others THEN
        plg_debug (SQLERRM);
    end;
end moduser;
/
show error

exit;
```

Binary Operations with Ldapadd

The `entryobj` object that the plug-in framework passes to an `ldapadd` plug-in now holds binary attributes as `binattr`. This variable is a table of `binattrobj` objects. The example that follows shows a post-add plug-in propagating a change (an added user) in the plug-in directory to another directory. In the latter directory, the entry appears under the DIT `cn=users,dc=us,dc=acme,dc=com`.

```
create or replace package adduser as
    procedure post_add(ldapplugincontext IN ODS.plugincontext,
        result IN integer,
        dn IN varchar2,
        entry IN ODS.entryobj,
        rc OUT integer,
        errormsg OUT varchar2);
end adduser;
/
show error

CREATE OR REPLACE PACKAGE BODY adduser AS
    procedure post_add(ldapplugincontext IN ODS.plugincontext,
        result IN integer,
        dn IN varchar2,
        entry IN ODS.entryobj,
        rc OUT integer,
        errormsg OUT varchar2)
    is
        counter1 pls_integer;
        counter2 pls_integer;
        retval pls_integer := -1;
        s integer;
        user_session DBMS_LDAP.session;
```

```

user_dn varchar(256);
user_array DBMS_LDAP.mod_array;
user_vals DBMS_LDAP.string_collection;
user_binvals DBMS_LDAP.blob_collection;
ldap_host varchar(256);
ldap_port varchar(256);
ldap_user varchar(256);
ldap_passwd varchar(256);
begin
  ldap_host := 'backup.us.oracle.com';
  ldap_port := '4000';
  ldap_user := 'cn=orcladmin';
  ldap_passwd := 'welcome';

  plg_debug('START ADDING THE ENTRY');

  -- Get a session
  user_session := dbms_ldap.init(ldap_host, ldap_port);

  -- Bind to the directory
  retval := dbms_ldap.simple_bind_s(user_session, ldap_user, ldap_passwd);

  -- Create a mod_array
  user_array := dbms_ldap.create_mod_array(entry.binattr.count +
  entry.attr.count);

  -- Create a user_dn
  user_dn := substr(dn,1,instr(dn,',',1,1))||'cn=users,dc=us,dc=acme,
  dc=com';
  plg_debug('THE CREATED DN IS' || user_dn);

  -- Populate the mod_array with binary attributes
  for counter1 in 1..entry.binattr.count loop
    for counter2 in 1..entry.binattr(counter1).binattrval.count loop
      plg_debug('THE NAME OF THE BINARY ATTR. IS' ||
      entry.binattr(counter1).binattrname);
      s := dbms_lob.getlength(entry.binattr(counter1).
      binattrval(counter2));
      plg_debug('THE NO. OF BYTES OF THE BINARY ATTR. VALUE IS' || s);
      user_binvals(counter2) := entry.binattr(counter1).
      binattrval(counter2);
    end loop;
    dbms_ldap.populate_mod_array(user_array,DBMS_LDAP.MOD_ADD,
    entry.binattr(counter1).binattrname,user_binvals);
    user_binvals.delete;
  end loop;

  -- Populate the mod_array with attributes
  for counter1 in 1..entry.attr.count loop
    for counter2 in 1..entry.attr(counter1).attrval.count loop
      plg_debug('THE NORMAL ATTRIBUTE' || entry.attr(counter1).attrname || '
      HAS THE VALUE' || entry.attr(counter1).attrval(counter2));
      user_vals(counter2) := entry.attr(counter1).attrval(counter2);
    end loop;
    dbms_ldap.populate_mod_array(user_array,DBMS_LDAP.MOD_ADD,
    entry.attr(counter1).attrname,user_vals);
    user_vals.delete;
  end loop;

  -- Add the entry

```

```

        retval := dbms_ldap.add_s(user_session,user_dn,user_array);
        plg_debug('THE RETURN VALUE IS' || retval);
        if retval = 0 then
            rc := 0;
            errormsg:='No error ocured while adding the entry';
        else
            rc := retval;
            errormsg :='Error code' || rc ||' while adding the entry';
        end if;

        -- Free the mod_array
        dbms_ldap.free_mod_array(user_array);
        retval := dbms_ldap.unbind_s(user_session);

        plg_debug('FINISHED ADDING THE ENTRY');

    exception
    WHEN others THEN
        plg_debug (SQLERRM);
    end;
end adduser;
/
show error

exit;

```

Binary Operations with ldapcompare

The `ldapcompare` plug-in can use three new overloaded module interfaces to compare binary attributes. If you want to use these interfaces to develop a plug-in package that handles both binary and nonbinary attributes, you must include two separate procedures in the package. The package name for both procedures is the same because only one `orclPluginName` can be registered in the plug-in entry.

After updating an existing plug-in package to include a procedure that compares binary attributes, reinstall the package. Recompile packages that depend on the plug-in package.

The three new interfaces look like this:

```

PROCEDURE pre_compare (ldapplugincontext IN ODS.plugincontext,
                      dn                 IN VARCHAR2,
                      attrname           IN VARCHAR2,
                      attrval           IN BLOB,
                      rc                 OUT INTEGER,
                      errormsg           OUT VARCHAR2 );

PROCEDURE when_compare_replace (ldapplugincontext IN ODS.plugincontext,
                                result           OUT INTEGER,
                                dn                 IN VARCHAR2,
                                attrname           IN VARCHAR2,
                                attrval           IN BLOB,
                                rc                 OUT INTEGER,
                                errormsg           OUT VARCHAR2 );

PROCEDURE post_compare (ldapplugincontext IN ODS.plugincontext,
                        result           IN INTEGER,
                        dn                 IN VARCHAR2,
                        attrname           IN VARCHAR2,
                        attrval           IN BLOB,
                        rc                 OUT INTEGER,

```

```
errormsg OUT VARCHAR2 );
```

The example that follows compares a binary attribute of an entry in the plug-in directory with a binary attribute of an entry in another directory. This package replaces the compare code of the server with the compare code of the plug-in. The package handles both binary and nonbinary attributes. As such it contains two separate procedures.

```
create or replace package compareattr as
  procedure when_compare_replace(ldapplugincontext IN ODS.plugincontext,
                                result OUT integer,
                                dn IN varchar2,
                                attrname IN VARCHAR2,
                                attrval IN BLOB,
                                rc OUT integer,
                                errormsg OUT varchar2);
  procedure when_compare_replace(ldapplugincontext IN ODS.plugincontext,
                                result OUT integer,
                                dn IN varchar2,
                                attrname IN VARCHAR2,
                                attrval IN varchar2,
                                rc OUT integer,
                                errormsg OUT varchar2);
end compareattr;
/
show error

CREATE OR REPLACE PACKAGE BODY compareattr AS
  procedure when_compare_replace(ldapplugincontext IN ODS.plugincontext,
                                result OUT integer,
                                dn IN varchar2,
                                attrname IN VARCHAR2,
                                attrval IN varchar2,
                                rc OUT integer,
                                errormsg OUT varchar2)
  is
    pos          INTEGER := 2147483647;
  begin
    plg_debug('START');
    plg_debug('THE ATTRNAME IS'||attrname||' AND THE VALUE IS'||attrval);
    plg_debug('END');
    rc := 0;
    errormsg := 'No error!!!';
  exception
  WHEN others THEN
    plg_debug ('Unknown UTL_FILE Error');
  end;

  procedure when_compare_replace(ldapplugincontext IN ODS.plugincontext,
                                result OUT integer,
                                dn IN varchar2,
                                attrname IN VARCHAR2,
                                attrval IN BLOB,
                                rc OUT integer,
                                errormsg OUT varchar2)
  is
    counter pls_integer;
    retval pls_integer := -1;
    cmp_result integer;
    s integer;
```

```

user_session DBMS_LDAP.session;
user_entry DBMS_LDAP.message;
user_message DBMS_LDAP.message;
user_dn varchar(256);
user_attrs DBMS_LDAP.string_collection;
user_attr_name VARCHAR2(256);
user_ber_elmt DBMS_LDAP.ber_element;
user_vals DBMS_LDAP.blob_collection;
ldap_host varchar(256);
ldap_port varchar(256);
ldap_user varchar(256);
ldap_passwd varchar(256);
ldap_base varchar(256);
begin
  ldap_host := 'backup.us.oracle.com';
  ldap_port := '4000';
  ldap_user := 'cn=orcladmin';
  ldap_passwd := 'welcome';
  ldap_base := dn;

  plg_debug('STARTING COMPARISON IN WHEN REPLACE PLUG-IN');

  s := dbms_lob.getlength(attrval);
  plg_debug('THE NUMBER OF BYTES OF ATTRVAL' || s);

  -- Get a session
  user_session := dbms_ldap.init(ldap_host, ldap_port);

  -- Bind to the directory
  retval := dbms_ldap.simple_bind_s(user_session, ldap_user, ldap_passwd);

  -- issue the search
  user_attrs(1) := attrname;
  retval := DBMS_LDAP.search_s(user_session, ldap_base,
                              DBMS_LDAP.SCOPE_BASE,
                              'objectclass=*',
                              user_attrs,
                              0,
                              user_message);

  -- Get the entry in the other OID server
  user_entry := DBMS_LDAP.first_entry(user_session, user_message);

  -- Log the DN and the Attribute name
  user_dn := DBMS_LDAP.get_dn(user_session, user_entry);
  plg_debug('THE DN IS' || user_dn);
  user_attr_name := DBMS_LDAP.first_attribute(user_session, user_entry,
  user_ber_elmt);

  -- Get the values of the attribute
  user_vals := DBMS_LDAP.get_values_blob(user_session, user_entry,
  user_attr_name);

  -- Start the binary comparison between the ATTRVAL and the attribute
  -- values
  if user_vals.count > 0 then
    for counter in user_vals.first..user_vals.last loop
      cmp_result := dbms_lob.compare(user_vals(counter), attrval,
  dbms_lob.getlength(user_vals(counter)), 1, 1);
      if cmp_result = 0 then

```



```

        rc := 0;
        -- Return LDAP_COMPARE_TRUE
        result := 6;
        plg_debug('THE LENGTH OF THE ATTR.'||user_attr_name||' IN THE
        ENTRY IS' ||dbms_lob.getlength(user_vals(counter)));
        errormsg := 'NO ERROR. THE COMPARISON HAS SUCCEEDED.';
        plg_debug(errormsg);
        plg_debug('FINISHED COMPARISON');
        return;
    end if;
end loop;
end if;

rc := 1;
-- Return LDAP_COMPARE_FALSE
result := 5;
errormsg := 'ERROR. THE COMPARISON HAS FAILED.';
plg_debug('THE LENGTH OF THE ATTR.'||user_attr_name||' IN THE ENTRY IS'
||dbms_lob.getlength(user_vals(user_vals.last)));
plg_debug(errormsg);
plg_debug('FINISHED COMPARISON');

-- Free user_vals
dbms_ldap.value_free_blob(user_vals);
exception
    WHEN others THEN
        plg_debug (SQLERRM);
    end;
end compareattr;
/
show error

exit;

```

Database Object Types Defined

This section defines the object types introduced in the Plug-in LDAP API. All of these definitions are in Oracle Directory Server database schema. Note that the API includes object types that enable plug-ins to extract binary data from the database.

```

create or replace type strCollection as TABLE of VARCHAR2(512);
/
create or replace type pluginContext as TABLE of VARCHAR2(512);
/
create or replace type attrvalType as TABLE OF VARCHAR2(4000);
/
create or replace type attrobj as object (
    attrname    varchar2(2000),
    attrval     attrvalType
);
/
create or replace type attrlist as table of attrobj;
/
create or replace type binattrvalType as TABLE OF BLOB;
/
create or replace type binattrobj as object (
    binattrname    varchar2(2000),
    binattrval     binattrvalType
);
/

```

```
create or replace type binattrlist as table of binattrobj;
/
create or replace type entryobj as object (
  entryname      varchar2(2000),
  attr           attrlist,
  binattr        binattrlist
);
/
create or replace type entrylist as table of entryobj;
/

create or replace type bvalobj as object (
  length         integer,
  val            varchar2(4000)
);
/
create or replace type bvallist as table of bvalobj;
/
create or replace type binvalobj as object (
  length         integer,
  binval         blob
);
/
create or replace type binvallist as table of binvalobj;
/
create or replace type modobj as object (
  operation      integer,
  type           varchar2(256),
  vals          bvallist,
  binvals        binvallist
);
/
create or replace type modlist as table of modobj;
```

Specifications for Plug-in Procedures

When you use the plug-ins, you must adhere to the signature defined for each of them. Each signature is provided here.

```
PROCEDURE pre_add (ldapplugincontext IN ODS.plugincontext,
  dn              IN VARCHAR2,
  entry           IN ODS.entryobj,
  rc              OUT INTEGER,
  errormsg       OUT VARCHAR2);
```

```
PROCEDURE when_add (ldapplugincontext IN ODS.plugincontext,
  dn              IN VARCHAR2,
  entry           IN ODS.entryobj,
  rc              OUT INTEGER,
  errormsg       OUT VARCHAR2);
```

```
PROCEDURE when_add_replace (ldapplugincontext IN ODS.plugincontext,
  dn              IN VARCHAR2,
  entry           IN ODS.entryobj,
  rc              OUT INTEGER,
  errormsg       OUT VARCHAR2);
```

```
PROCEDURE post_add (ldapplugincontext IN ODS.plugincontext,
result          IN INTEGER,
dn              IN VARCHAR2,
entry          IN ODS.entryobj,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2);

PROCEDURE pre_modify (ldapplugincontext IN ODS.plugincontext,
dn              IN VARCHAR2,
mods           IN ODS.modlist,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2);

PROCEDURE when_modify (ldapplugincontext IN ODS.plugincontext,
dn              IN VARCHAR2,
mods           IN ODS.modlist,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2);

PROCEDURE when_modify_replace (ldapplugincontext IN ODS.plugincontext,
dn              IN VARCHAR2,
mods           IN ODS.modlist,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2);

PROCEDURE post_modify (ldapplugincontext IN ODS.plugincontext,
result          IN INTEGER,
dn              IN VARCHAR2,
mods           IN ODS.modlist,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2);

PROCEDURE pre_compare (ldapplugincontext IN ODS.plugincontext,
dn              IN VARCHAR2,
attrname       IN VARCHAR2,
attrval        IN VARCHAR2,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2
);

PROCEDURE pre_compare (ldapplugincontext IN ODS.plugincontext,
dn              IN VARCHAR2,
attrname       IN VARCHAR2,
attrval        IN BLOB,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2 );

PROCEDURE when_compare_replace (ldapplugincontext IN ODS.plugincontext,
result          OUT INTEGER,
dn              IN VARCHAR2,
attrname       IN VARCHAR2,
attrval        IN VARCHAR2,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2
);
```

```
PROCEDURE when_compare_replace (ldapplugincontext IN ODS.plugincontext,
result          OUT INTEGER,
dn              IN VARCHAR2,
attrname       IN VARCHAR2,
attrval        IN BLOB,
rc              OUT INTEGER,
errormsg       OUT VARCHAR2 );
```

```
PROCEDURE post_compare (ldapplugincontext IN ODS.plugincontext,
result          IN INTEGER,
dn              IN VARCHAR2,
attrname       IN VARCHAR2,
attrval        IN VARCHAR2,
rc              OUT INTEGER,
errormsg       OUT VARCHAR2
);
```

```
PROCEDURE post_compare (ldapplugincontext IN ODS.plugincontext,
result          IN INTEGER,
dn              IN VARCHAR2,
attrname       IN VARCHAR2,
attrval        IN BLOB,
rc              OUT INTEGER,
errormsg       OUT VARCHAR2 );
```

```
PROCEDURE pre_delete (ldapplugincontext IN ODS.plugincontext,
dn              IN VARCHAR2,
rc              OUT INTEGER,
errormsg       OUT VARCHAR2
);
```

```
PROCEDURE when_delete (ldapplugincontext IN ODS.plugincontext,
dn              IN VARCHAR2,
rc              OUT INTEGER,
errormsg       OUT VARCHAR2
);
```

```
PROCEDURE when_delete_replace (ldapplugincontext IN ODS.plugincontext,
dn              IN VARCHAR2,
rc              OUT INTEGER,
errormsg       OUT VARCHAR2
);
```

```
PROCEDURE post_delete (ldapplugincontext IN ODS.plugincontext,
result          IN INTEGER,
dn              IN VARCHAR2,
rc              OUT INTEGER,
errormsg       OUT VARCHAR2
);
```

```
PROCEDURE pre_search (ldapplugincontext IN ODS.plugincontext,
baseDN         IN VARCHAR2,
scope          IN INTEGER,
filterStr      IN VARCHAR2,
requiredAttr   IN ODS.strCollection,
rc              OUT INTEGER,
errormsg       OUT VARCHAR2
);
```

```
PROCEDURE post_search (ldapplugincontext IN ODS.plugincontext,
result         IN INTEGER,
```

```
basedN      IN VARCHAR2,  
scope       IN INTEGER,  
filterStr   IN VARCHAR2,  
requiredAttr IN ODS.strCollection,  
rc          OUT INTEGER,  
errmsg      OUT VARCHAR2  
);
```

```
PROCEDURE pre_bind (ldapplugincontext IN ODS.plugincontext,  
dn          IN VARCHAR2,  
passwd      IN VARCHAR2,  
rc          OUT INTEGER,  
errmsg      OUT VARCHAR2  
);
```

```
PROCEDURE when_bind_replace (ldapplugincontext IN ODS.plugincontext,  
result      OUT INTEGER,  
dn          IN VARCHAR2,  
passwd      IN VARCHAR2,  
rc          OUT INTEGER,  
errmsg      OUT VARCHAR2  
);
```

```
PROCEDURE post_bind (ldapplugincontext IN ODS.plugincontext,  
result      IN INTEGER,  
dn          IN VARCHAR2,  
passwd      IN VARCHAR2,  
rc          OUT INTEGER,  
errmsg      OUT VARCHAR2  
);
```

Integrating with Oracle Delegated Administration Services

This chapter explains how to integrate applications with Oracle Delegated Administration Services. This Web tool enables you to more easily develop tools for administering application data in the directory.

It contains the following sections:

- [What Is Oracle Delegated Administration Services?](#)
- [Integrating Applications with the Delegated Administration Services](#)
- [Java APIs Used to Access URLs](#)

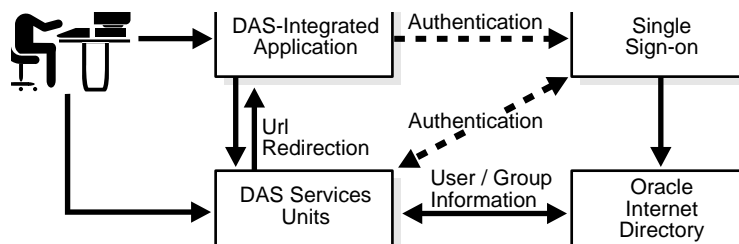
What Is Oracle Delegated Administration Services?

Oracle Delegated Administration Services consists of a set of pre-defined, Web-based service units for performing directory operations on behalf of users. These units enable directory users to update their own information.

The delegated administration services provide most of the functionality that directory-enabled applications require. You can use the service units to create user and group entries, search for entries, and change user passwords.

You can embed delegated administration service units in your applications. If, for example, you are building a Web portal, you can add service units that enable users to change application passwords stored in the directory. Each service unit has a corresponding URL stored in the directory. At runtime, an application can find the URL by querying the directory.

Figure 8-1 Overview of Delegated Administration Services



How Applications Benefit from Oracle Delegated Administration Services

An application based on Oracle Delegated Administration Services is more advanced than one based on earlier types of APIs. First, an application developed using the service units is language independent because the units are Web based. This means that the application can handle input and requests from any type of user or application, eliminating the need for a costly custom solution or configuration. Second, Oracle Delegated Administration Services comes with the Oracle Internet Directory Self-Service Console, a GUI development tool that automates many of the directory-oriented application requirements (such as Create, Edit, and Delete). Third, Oracle Delegated Administration Services is integrated with Oracle Application Server Single Sign-On. The application is automatically authenticated by the single sign-on server. This means that the application can query the directory on a user's behalf.

Integrating Applications with the Delegated Administration Services

This section contains these topics:

- [Integration Profile](#)
- [Integration Methodology and Considerations](#)

Integration Profile

An application integrated with Oracle Delegated Administration Services has the following characteristics:

- It is a Web-based GUI.
- It is integrated with Oracle Application Server Single Sign-On through mod_osso.
- It has operations that it must perform by way of a signed-on user. It can perform these operations using Oracle Delegated Administration Services.
- It has users or groups stored in Oracle Internet Directory and can use Oracle Delegated Administration Services for user and group management.
- It runs on the Oracle Application Server infrastructure or middle-tier. The discovery mechanism for the service URLs is inaccessible otherwise.

Integration Methodology and Considerations

[Table 8-1](#) on page 8-2 identifies the tasks that are required to integrate an application with Oracle Delegated Administration Services.

Table 8-1 *Integration Considerations*

Point in Application Lifecycle	Considerations
Application design time	<p>Examine the various services that Oracle Delegated Administration Services provides. Identify integration points within the application GUI.</p> <p>Make code changes to pass parameters to the Oracle Delegated Administration Services self-service units and to process return parameters from Oracle Delegated Administration Services.</p> <p>Introduce code in the bootstrap and installation logic to dynamically discover the location of Oracle Delegated Administration Services units from configuration information in Oracle Internet Directory. To do this, use Oracle Internet Directory Service Discovery APIs.</p>

Table 8–1 (Cont.) Integration Considerations

Point in Application Lifecycle	Considerations
Application installation time	Determine the location of Oracle Delegated Administration Services units and store them in local repository.
Application runtime	<p>Display Oracle Delegated Administration Services URLs in application GUI shown to users.</p> <p>Pass the appropriate parameters to the Oracle Delegated Administration Services by using URL encoding.</p> <p>Process return codes from Oracle Delegated Administration Services through the URL return.</p>
Ongoing administrative activities	Provide the capability to refresh the location of Oracle Delegated Administration Services and its URLs in the administrator screens. Do this in case the deployment moves the location of Oracle Delegated Administration Services after the application has been installed.

Use Case 1: Create User

This use case shows how to integrate the Create User unit with a custom application. In the custom application page, Create User is shown as a link.

1. Identify the base URL for Oracle Delegated Administration Services by using this Java API string:

```
baseUrl = Util.getDASUrl(ctx,DASURL_BASE)
```

This API returns the base URL in this form: `http://host_name:port/`

2. Get the URL for the Create User unit by using this string:

```
relUrl = Util.getDASUrl ( ctx , DASURL_CREATE_USER )
```

The return value is the relative URL to access the Create User unit.

The specific URL is the information needed to generate the link dynamically for the application.

You can customize the parameters in [Table 8–2](#) on page 8-3 for this unit.

Table 8–2 URL Parameters for Oracle Delegated Administration Services

Parameter	Description
homeURL	The URL that is linked to the global button Home in the Oracle Delegated Administration Services unit. When the calling application specifies this value, you can click Home to redirect the Oracle Delegated Administration Services unit to the URL specified by this parameter.
doneURL	This URL is used by Oracle Delegated Administration Services to redirect the Oracle Delegated Administration Services page at the end of each operation. In the case of Create User, once the user is created, clicking OK redirects the URL to this location.
cancelURL	This URL is linked with all the Cancel buttons shown in Oracle Delegated Administration Services units. Any time the user clicks Cancel, the page is redirected to the URL specified by this parameter.
enablePA	This parameter takes a Boolean value of true or false. This will enable the Assign Privileges section in a User or Group operation. If <code>enablePA</code> is passed with value of true in the Create User page, then the Assign Privileges to User section will also appear on the Create User Page.

3. Build the link with the parameters set to the following values:

```
baseUrl = http://acme.mydomain.com:7777/  
relUrl = oiddas/ui/oracle/ldap/das/admin/AppCreateUserInfoAdmin  
homeURL = http://acme.mydomain.com/myapp  
cancelURL = http://acme.mydomain.com/myapp  
doneURL = http://acme.mydomain.com/myapp  
enablePA = true
```

The complete URL looks like this:

```
http://acme.mydomain.com:7777/oiddas/ui/oracle/ldap/das/admin/  
AppCreateUserInfoAdmin?homeURL=http://acme.mydomain.com/myapp&  
cancelURL=http://acme.mydomain.com/myapp&  
doneURL=http://acme.mydomain.com/myapp&  
enablePA=true
```

4. You can now embed this URL in the application.**Use Case 2: User LOV**

List of Values (LOV) is implemented using JavaScript to invoke and pass values between the LOV calling window and the LOV page. The application invoking the LOV needs to open a popup window using JavaScript. Because Java scripts have security restrictions, no data may cross domains. Due to this limitation, only pages in the same domain can access the LOV units.

Base and relative URLs can be invoked the same way as they are for Create User.

Sample files are located at:

```
$ORACLE_HOME/ldap/das/samples/lov
```

The samples illustrate how the LOV can be invoked and data can be passed between the calling application and the Oracle Delegated Administration Services unit. A Complete illustration of the LOV invocation is beyond the scope of this chapter.

Java APIs Used to Access URLs

Java APIs can be used to discover URLs for Oracle Delegated Administration Services. More details about these APIs are provided in [Chapter 3, "Developing Applications with Oracle Extensions to the Standard APIs"](#) and in [Chapter 15, "DAS_URL Interface Reference"](#). The API functions that address URL discovery are `getDASUrl(DirContext ctx, String urlTypeDN)` and `getAllDASUrl(DirContext ctx)`.

Developing Applications for Single Sign-On

This chapter explains how to develop applications to work with `mod_osso`. The chapter contains the following topics:

- [What Is `mod_osso`?](#)
- [Protecting Applications Using `mod_osso`: Two Methods](#)
- [Developing Applications Using `mod_osso`](#)
- [Security Issues](#)

What Is `mod_osso`?

In OracleAS release 10.1.2, you use `mod_osso`, an authentication module on the Oracle HTTP Server, to enable applications for single sign-on. `mod_osso` is a simple alternative to the single sign-on SDK, used in earlier releases to integrate partner applications. `mod_osso` simplifies the authentication process by serving as the sole partner application to the single sign-on server. By doing so, it renders authentication transparent for OracleAS applications.

After authenticating users, `mod_osso` transmits the simple header values that applications need to validate them. These include the following:

- User name
- User GUID
- Language and territory

[Table 9-1](#) lists all of the user attributes that `mod_osso` passes to applications. The table also recommends attributes to use as keys, or handles, to retrieve additional user attributes from Oracle Internet Directory.

Table 9-1 *User Attributes Passed to Partner Applications*

HTTP Header Name	Description	Source	Use as Key or Handle?
<code>Ossso-User-Guid</code>	Single sign-on user's globally unique user ID (GUID).	Single sign-on user's globally unique user ID (GUID).	Recommended.
<code>Ossso-Subscriber-Guid</code>	Realm GUID.	Realm entry in Oracle Internet Directory.	Recommended.

Table 9–1 (Cont.) User Attributes Passed to Partner Applications

HTTP Header Name	Description	Source	Use as Key or Handle?
Remote-User	User nickname as entered by user on the login page.	Single sign-on login page.	Recommended for pre-9.0.4 applications only.
Ossso-Subscriber	User-friendly name for a realm.	Realm entry in Oracle Internet Directory.	Not recommended. Use GUID headers to perform user searches in Oracle Internet Directory.
Accept-Language	Language and territory in ISO format.	Single sign-on server.	Not applicable.

mod_osso interoperates only with the Oracle HTTP listener. You can use OracleAS SSO Plug-in to protect applications that work with third-party listeners such as Sun One and IIS. To learn how to use OracleAS SSO Plug-in, see the appendix about this tool in *Oracle HTTP Server Administrator's Guide*.

Protecting Applications Using mod_osso: Two Methods

mod_osso redirects the user to the single sign-on server only if the URL you request is configured to be protected. You can secure URLs in one of two ways: statically or dynamically. Static directives simply protect the application, ceding control over user interaction to mod_osso. Dynamic directives not only protect the application, they also enable it to regulate user access.

This section contains the following topics:

- [Protecting URLs Statically](#)
- [Protecting URLs with Dynamic Directives](#)

Protecting URLs Statically

You can statically protect URLs with mod_osso by applying directives to the mod_osso.conf file. This file is found at \$ORACLE_HOME/Apache/Apache/conf. In the example that follows, a directory named /private, located just below the Oracle HTTP Server document root, is protected by this directive:

```
<IfModule mod_osso.c>

    <Location /private>
        AuthType Basic
        require valid-user
    </Location>

</IfModule>
```

After making the entry, restart the Oracle HTTP Server:

```
$ORACLE_HOME/opmn/bin/opmnctl restartproc type=ohs
```

Finally, populate the directory with pages and then test them. For example:

```
http://host:port/private/helloworld.html
```

Protecting URLs with Dynamic Directives

Dynamic directives are HTTP response headers that have special error codes that enable an application to request granular functionality from the single sign-on system

without having to implement the intricacies of the single sign-on protocol. Upon receiving a directive as part of a simple HTTP response from the application, mod_osso creates the appropriate single sign-on protocol message and communicates it to the single sign-on server.

OracleAS supports dynamic directives for Java servlets and JSPs. The product does not currently support dynamic directives for PL/SQL applications.

[Table 9–2](#) lists commonly requested dynamic directives.

Table 9–2 Commonly Requested Dynamic Directives

Directive	Status Code	Headers
Request Authentication	401, 499	-
Request Forced Authentication	499	Osso-Paranoid: true
Single Sign-Off	470	Osso-Return-URL This is the URL to return to after single sign-off is complete

Developing Applications Using mod_osso

This section explains how to write and enable applications using mod_osso. The section contains the following topics:

- [Developing Statically Protected PL/SQL Applications](#)
- [Developing Statically Protected Java Applications](#)
- [Developing Java Applications That Use Dynamic Directives](#)
- [A Word About Non-GET Authentication](#)

Developing Statically Protected PL/SQL Applications

What follows is an example of a simple mod_osso-protected application. This application logs the user in to the single sign-on server, displays user information, and then logs the user out of both the application and the single sign-on server.

Use the following steps to write and enable a PL/SQL application using mod_osso.

1. Create the schema where application procedure will be loaded.

```
sqlplus sys/sys_password as sysdba
create user schema_name identified by schema_password;
grant connect, resource to schema_name;
```

2. Load the following procedure into the schema and grant the public access to the procedure:

```
create or replace procedure show_user_info
is
begin
begin
    http.init;
exception
    when others then null;
end;
http.htmlOpen;
http.bodyOpen;
http.print('<h2>Welcome to Oracle Single Sign-On</h2>');
```

```

        http.print('<pre>');
        http.print('Remote user:'
            || owa_util.get_cgi_env('REMOTE_USER'));
        http.print('User DN:'
            || owa_util.get_cgi_env('Osso-User-Dn'));
        http.print('User Guid:'
            || owa_util.get_cgi_env('Osso-User-Guid'));
        http.print('Subscriber:'
            || owa_util.get_cgi_env('Osso-Subscriber'));
        http.print('Subscriber DN:'
            || owa_util.get_cgi_env('Osso-Subscriber-Dn'));
        http.print('Subscriber Guid:'
            || owa_util.get_cgi_env('Osso-Subscriber-Guid'));
        http.print('</pre>');
        http.print('<a href=/osso_logout?'
            || 'p_done_url=http://my.oracle.com>Logout</a>');

        http.bodyClose;
        http.htmlClose;
    end show_user_info;
/
show errors;

grant execute on show_user_info to public;

```

3. Create a database access descriptor (DAD) for the application in the `dads.conf` file, located at `$ORACLE_HOME/Apache/modplsql/conf`:

```

<Location /pls/DAD_name>
    SetHandler pls_handler
    Order deny,allow
    AllowOverride None
    PlsqlDatabaseConnectString    hostname:port:SID
    PlsqlDatabasePassword         schema_password
    PlsqlDatabaseUsername         schema_name
    PlsqlDefaultPage              schema_name.show_user_info
    PlsqlDocumentTablename        schema_name.wwdoc_document
    PlsqlDocumentPath             docs
    PlsqlDocumentProcedure        schema_name.wwdoc_process.process_
                                   download
    PlsqlAuthenticationMode       Basic
    PlsqlPathAlias                 url
    PlsqlPathAliasProcedure        schema_name.wwpth_api_alias.process_
                                   download
    PlsqlSessionCookieName        schema_name
    PlsqlCGIEnvironmentList        OSSO-USER-DN
    PlsqlCGIEnvironmentList        OSSO-USER-GUID
    PlsqlCGIEnvironmentList        OSSO-SUBSCRIBER
    PlsqlCGIEnvironmentList        OSSO-SUBSCRIBER-DN
    PlsqlCGIEnvironmentList        OSSO-SUBSCRIBER-GUID
</Location>

```

4. Protect the application DAD by entering the following lines in the `mod_osso.conf` file:

```

<Location /pls/DAD_name>
    require valid-user
    authType Basic
</Location>

```

Note: The assumption here is that mod_osso is already configured for single sign-on. This step is performed when OracleAS is installed.

5. Restart the Oracle HTTP Server:

```
http://host:port/private/helloworld.html
```

6. To test whether the newly created functions and procedures are protected by mod_osso, try to access them from a browser:

```
http://host:port/pls/DAD/schema_name.show_user_info
```

Selecting the URL should invoke the single sign-on login page if mod_osso.conf has been configured properly and mod_osso is registered with the single sign-on server.

Developing Statically Protected Java Applications

Use the following steps to write and enable a servlet or JSP application using mod_osso:

1. Write the JSP or servlet. Like the PL/SQL application example immediately preceding, the simple servlet that follows logs the user in, displays user information, and then logs the user out.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Example servlet showing how to get the SSO User information
 */

public class SSOProtected extends HttpServlet
{

    public void service(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");

        // Show authenticated user informationsingle sign-on
        PrintWriter out = response.getWriter();
        out.println("<h2>Welcome to Oracle Single Sign-On</h2>");
        out.println("<pre>");
        out.println("Remote user: "
            + request.getRemoteUser());
        out.println("Osso-User-Dn: "
            + request.getHeader("Osso-User-Dn"));
        out.println("Osso-User-Guid: "
            + request.getHeader("Osso-User-Guid"));
        out.println("Osso-Subscriber: "
            + request.getHeader("Osso-Subscriber"));
        out.println("Osso-User-Dn: "
            + request.getHeader("Osso-User-Dn"));
        out.println("Osso-Subscriber-Dn: "
            + request.getHeader("Osso-Subscriber-Dn"));
        out.println("Osso-Subscriber-Guid: "
```

```

        + request.getHeader("Osso-Subscriber-Guid"));
    out.println("Lang/Territory: "
        + request.getHeader("Accept-Language"));
    out.println("</pre>");
    out.println("<a href=/osso_logout?"
        +"p_done_url=http://my.oracle.com>Logout</a>");

```

2. Protect the servlet by entering the following lines in the `mod_osso.conf` file:

```

<Location /servlet>
    require valid-user
    authType Basic
</Location>

```

3. Deploy the servlet. If you need help, see the overview chapter in *Oracle Application Server Containers for J2EE Servlet Developer's Guide*. This chapter provides an example of a servlet and shows how to deploy it.
4. Restart the Oracle HTTP Server and OC4J:

```

$ORACLE_HOME/opmn/bin/opmnctl restartproc type=ohs
$ORACLE_HOME/opmn/bin/opmnctl stopproc type=oc4j
$ORACLE_HOME/opmn/bin/opmnctl startproc type=oc4j

```

5. Test the servlet by trying to access it from the browser. Selecting the URL should invoke the login page.

The process is this: when you try to access the servlet from the browser, you are redirected to the single sign-on server for authentication. Next you are redirected back to the servlet, which displays user information. You may then select the logout link to log out of the application as well as the single sign-on server.

Developing Java Applications That Use Dynamic Directives

Applications that use dynamic directives require no entry in `mod_osso.conf` because `mod_osso` protection is written directly into the application as one or more dynamic directives. The servlets that follow show how such directives are incorporated. Like their "static" counterparts, these sample "dynamic" applications generate user information.

This section covers the following topics:

- Java Example #1: Simple Authentication
- Java Example #2: Single Sign-Off
- Java Example #3: Forced Authentication

Java Example #1: Simple Authentication

This servlet uses the `request.getRemoteUser()` method to check the `mod_osso` cookie for the user name. If the user name is absent, the servlet issues dynamic directive 499, a request for simple authentication. The key lines are in boldface.

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Example servlet showing how to use
 * Dynamic Directive for login
 */

```



```

public class SSODynLogin extends HttpServlet
{
    public void service(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException
    {
        String l_user    = null;

        // Try to get the authenticate user name
        try
        {
            l_user = request.getRemoteUser();
        }
        catch(Exception e)
        {
            l_user = null;
        }

        // If user is not authenticated then generate
        // dynamic directive for authentication
        if((l_user == null) || (l_user.length() <= 0) )
        {
            response.sendError(499, "Oracle SSO");
        }
        else
        {
            // Show authenticated user information
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            out.println("<h2>Welcome to Oracle Single Sign-On</h2>");
            out.println("<pre>");
            out.println("Remote user: "
                + request.getRemoteUser());
            out.println("Osso-User-Dn: "
                + request.getHeader("Osso-User-Dn"));
            out.println("Osso-User-Guid: "
                + request.getHeader("Osso-User-Guid"));
            out.println("Osso-Subscriber: "
                + request.getHeader("Osso-Subscriber"));
            out.println("Osso-User-Dn: "
                + request.getHeader("Osso-User-Dn"));
            out.println("Osso-Subscriber-Dn: "
                + request.getHeader("Osso-Subscriber-Dn"));
            out.println("Osso-Subscriber-Guid: "
                + request.getHeader("Osso-Subscriber-Guid"));
            out.println("Lang/Territory: "
                + request.getHeader("Accept-Language"));
            out.println("</pre>");
        }
    }
}

```

Note: If Oracle JAAS Provider is used, the directive code 401 may be substituted for 499.

Java Example #2: Single Sign-Off

This servlet is invoked when users select the login link within an application. The application sets the URL to return to when sign-off is complete; then it issues a directive that sends users to the single sign-off page. The key lines are in boldface.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Example servlet showing how to use
 * Dynamic Directive for logout
 */

public class SSODynLogout extends HttpServlet
{
    public void service (HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        // Set the return URL
        response.setHeader("Osso-Return-Url",
            "http://my.oracle.com" );
        // Send Dynamic Directive for logout
        response.sendError(470, "Oracle SSO");
    }
}
```

Note: Alternatively, you can redirect to the `osso_logout` URL on that computer.

Java Example #3: Forced Authentication

If logged-in users have exceeded a timeout, an application can force them to reauthenticate. The directive for reauthentication is written into the servlet that follows. The key lines are in boldface.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Example servlet showing how to use
 * Dynamic Directive for forced login
 */

public class SSODynForcedLogin extends HttpServlet
{
    public void service(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException
    {
        String l_user = null;
        // Try to get the authenticate user name
        try
        {
            l_user = request.getRemoteUser();
        }
    }
}
```

```

catch(Exception e)
{
    l_user = null;
}

// If the user is authenticated then show
// user information; otherwise generate Dynamic
// Directive for forced login
if(l_user != null)
{
    // Show authenticated user information
    PrintWriter out = response.getWriter();
    response.setContentType("text/html");
    out.println("<h2>Welcome to Oracle Single Sign-On.</h2>");
    out.println("<pre>");
    out.println("Remote user: "
        + request.getRemoteUser());
    out.println("Osso-User-Dn: "
        + request.getHeader("Osso-User-Dn"));
    out.println("Osso-User-Guid: "
        + request.getHeader("Osso-User-Guid"));
    out.println("Osso-Subscriber: "
        + request.getHeader("Osso-Subscriber"));
    out.println("Osso-User-Dn: "
        + request.getHeader("Osso-User-Dn"));
    out.println("Osso-Subscriber-Dn: "
        + request.getHeader("Osso-Subscriber-Dn"));
    out.println("Osso-Subscriber-Guid: "
        + request.getHeader("Osso-Subscriber-Guid"));
    out.println("Lang/Territory: "
        + request.getHeader("Accept-Language"));
    out.println("</pre>");
}
else
{
    response.setHeader("Osso-Paranoid", "true");
    response.sendError(499, "Oracle SSO");
}
}
}

```

A Word About Non-GET Authentication

The first page of a mod_osso-protected application must be a URL that uses the GET authentication method. If the POST method is used, the data that the user provides when logging in is lost during redirection to the single sign-on server. When deciding whether to enable the global user inactivity timeout, please note that users are redirected after timing out and logging in again.

Global Inactivity Timeout and Dynamic Directives

If you are using Global Inactivity Timeout and Dynamic Directive for enabling Single Sign-On for your applications, then you can use the `Osso-Idle-Timeout-Exceeded` HTTP header in your application to determine the timeout status. This header value is set to `true` if timeout has occurred, otherwise it is set to `false`.

The following example shows how you can use the `Osso-Idle-Timeout-Exceeded` HTTP header:

```
// Get the timeout status
String timeoutStatus = request.getHeader("Osso-Idle-Timeout-Exceeded")
// Check if user has timedout
if ( (timeoutStatus != null) && timeoutStatus.equalsIgnoreCase("true") )
{
    response.setHeader( "Osso-Paranoid", "true" );
    response.sendError(499, "Oracle SSO");
}
else
{
    // Display page content here
}
```

Security Issues

This section describes security considerations when developing applications for OracleAS Single Sign-On. It contains these topics:

- [Single Sign-Off and Application Logout](#)
- [Secure Transmission of mod_osso Cookies](#)

Single Sign-Off and Application Logout

If you build custom applications using OracleAS, note the following: when global logout, or single sign-off, is invoked, only the single sign-on and mod_osso cookies are cleared. This means that an OracleAS application must be coded to store single sign-on user and realm names in either the OC4J session or in the application session. The application must then compare these values to those passed by mod_osso. If a match occurs, the application must show personalized content. If no match occurs, which means that the mod_osso cookie is absent, the application must clear the application session and force the user to log in.

This section covers the following topics:

- [Application Login: Code Examples](#)
- [Application Logout: Recommended Code](#)

Application Login: Code Examples

The first two code examples in this section do not incorporate the logic prescribed in the section immediately preceding. The third example does incorporate this logic. Although these are Java examples, they could be examples written in other languages such as Perl, PL/SQL, and CGI.

Bad Code Example #1

```
// Get user name from application session. This session was
// established by the application cookie or OC4J session cookie
String username = request.getSession().getAttribute('USER_NAME');

// Get subscriber name from application session. This session was
// established by the application cookie or OC4J session cookie.
String subscriber = request.getSession().getAttribute('SUBSCRIBER_NAME');

// Get user security information from application session. This session was
// established by the application cookie or OC4J session cookie
String user_sec_info = request.getSession().getAttribute('USER_APP_SEC');

if((username != null) && (subscriber!= null))
```

```

{
    // Show personalized user content
    show_personalized_page(username, subscriber, user_sec_info);
}
else
{
    // Send Dynamic Directive for login
    response.sendError( 499, "Oracle SSO" );
}

```

Bad Code Example #2

```

// Get SSO username from http header
String username = request.getRemoteUser();

// Get subscriber name from SSO http header
String subscriber = request.getHeader('OSSO-SUBSCRIBER');

// Get user security information from application session.
// This session was established by the application or OC4J session.
String user_sec_info = request.getSession().getAttribute('USER_APP_SEC');

if((ssousername != null)&&(subscriber!= null))
{
    // Show personalized user content
    show_personalized_page(username, subscriber, user_sec_info);
}
else
{
    // Send Dynamic Directive for login
    response.sendError( 499, "Oracle SSO" );
}

```

Recommended Code

```

// Get user name from application session. This session was
// established by the application or OC4J session
String username = request.getSession().getAttribute('USER_NAME');

// Get subscriber name from application session. This session was
// established by the application or OC4J session
String subscriber = request.getSession().getAttribute('SUBSCRIBER_NAME');

// Get user security information from application session.
// This session was established by the application or OC4J session.
String user_sec_info = request.getSession().getAttribute('USER_APP_SEC');

// Get username and subscriber name from JAZN API */
JAZNUserAdaptor jaznuser = (JAZNUserAdaptor)request.getUserPrincipal();
String ssousername = jaznuser.getName();
String ssosubscriber = jaznuser.getRealm().getName();

// If you are not using JAZN api then you can also get the username and
// subscriber name from mod_osso headers
String ssousername = request.getRemoteUser();
String ssosubscriber = request.getHeader('OSSO-SUBSCRIBER');

// Check for application session. Create it if necessary.
if((username == null) || (subscriber == null) )
{
    ...Code to create application session. Get the user information from
    JAZN api (or mod_osso headers if you are not using JAZN api) and populate the
}

```

```
        application session with user, subscriber, and user security info.
    }

    if((username != null)&&(subscriber != null)
        &&(ssouusername != null)&&(ssosubscriber != null)
        &&(username.equalsIgnoreCase(ssouusername) == 0 )
        &&(subscriber.equalsIgnoreCase(ssosubscriber) == 0))
    {
        // Show personalized user content
        show_personalized_page(username, subscriber, user_sec_info);
    }
    else
    {
        ...Code to Wipe-out application session, followed by...

        // Send Dynamic Directive for login
        // If you are using JAZN then you should use following code
        // response.sendError( 401);

        // If you are not using JAZN api then you should use following code
        // response.sendError( 499, "Oracle SSO" );
    }
}
```

Application Logout: Recommended Code

Most applications that authenticate users have a logout link. In a single-sign-on-enabled application, the user invokes the dynamic directive for logout in addition to other code in the logout handler of the application. Invoking the logout directive initiates single sign-off, or global logout. The example that follows shows what single sign-off code should look like in Java:

```
// Clear application session, if any
String l_return_url := return url to your application
response.setHeader( "Osso-Return-Url", l_return_url);
response.sendError( 470, "Oracle SSO" );
```

Secure Transmission of mod_osso Cookies

You can add the `OssosSecureCookies` directive to set the `Secure` flag on all cookies created by `mod_osso`. This tells the browser to only transmit those cookies on connections secured by HTTPS.

An example of this directive, in the `mod_osso` configuration file located in `$ORACLE_HOME/Apache/Apache/conf/mod_osso.conf`, is as follows:

```
<IfModule mod_osso.c>
    OossoIpCheck off
    OossoIdleTimeout off
    OossoSecureCookies on
    OossoConfigFile osso/osso.conf

    <Location /j2ee/webapp>
        require valid-user
        AuthType Basic
    </Location>

</IfModule>
```

Integrating J2EE Applications and Oracle Internet Directory

This chapter is designed to provide a short overview of APIs you can use in J2EE applications to get information about user permissions, groups, and policies from Oracle Internet Directory.

Oracle Application Server Containers for J2EE (OC4J) is a J2EE certified server implementation. OC4J supports the standard J2EE security APIs.

In addition to the standard security APIs, OC4J provides a set of security features collectively known as JAZN. JAZN includes the Oracle Application Server Java Authentication and Authorization Service (JAAS) Provider, the JAZN User Manager, the JAAS Policy Management API, and the Realm API. OC4J is fully integrated with Oracle Application Server Single Sign-On and Oracle Internet Directory. JAZN security APIs provide features not found in standard J2EE security APIs.

The OracleAS JAAS Provider is an implementation of Java Authentication and Authorization Services (JAAS) that stores security policies in either XML files or in Oracle Internet Directory. OC4J applications can use JAAS Policy Management APIs for fine-grained authorization.

This document discusses the following topics:

- [Standard J2EE Security APIs](#)
- [OC4J Security APIs](#)
- [JAAS Policy Management APIs](#)

Standard J2EE Security APIs

The J2EE standard implementation includes security APIs that can be used by Java Servlets and Enterprise JavaBeans (EJBs) to get information about users and roles. These APIs work independently from Oracle Internet Directory. They retrieve information about users who have already been authenticated, regardless of whether the application is integrated with Oracle Identity Management.

The `javax.servlet.http` package, which is part of the Java Servlet specification, includes the following methods for obtaining information about users:

- `javax.servlet.http.HttpServletRequest.getUserPrincipal()`
- `javax.servlet.http.HttpServletRequest.isUserInRole()`
- `javax.servlet.http.HttpServletRequest.getRemoteUser()`

To learn more about the `javax.servlet.http` package, see:

<http://java.sun.com/products/servlet/2.2/javadoc/index.html>

Similarly, the `javax.ejb` package, which is part of the Enterprise JavaBeans specification, includes the following methods for obtaining information about users:

- `javax.ejb.EJBContext.getCallerPrincipal()`
- `javax.ejb.EJBContext.isCallerInRole()`

To learn more about the `javax.ejb` package, see:

<http://java.sun.com/j2ee/1.4/docs/api/javax/ejb/package-tree.html>

OC4J Security APIs

JAZN security APIs are based on the package `com.evermind.security`. This class specifies a user manager to authenticate and authorize users and groups that attempt to access a J2EE application. The default JAZN user manager is `JAZNUserManager`, which supports LDAP-based providers and is integrated with Oracle Application Server Single Sign-On and Oracle Internet Directory.

To access Oracle Internet Directory information using `JAZNUserManager`, you must configure JAZN to use the LDAP-based provider, `jazn-ldap`, as described in the *Oracle Application Server Containers for J2EE Security Guide*.

JAZN supports the following `com.evermind.security.User` methods to retrieve user attributes from Oracle Internet Directory:

- `getDescription()` returns a short description of this user or null if no description is present.
- `getGroups()` returns the groups that this user belongs to, if known and supported.
- `getName()` returns the username of this user.
- `hasPermission()` checks whether this user has the named permission.
- `isMemberOf()` checks whether this user is a member of the specified group.

See *JAAS Provider API Reference* for more information.

Applications that need additional user attributes, such as email address or Oracle Internet Directory-specific attributes, must use the Oracle Internet Directory APIs. These are found in *Oracle Internet Directory API Reference* and discussed in [Chapter 2](#) and [Chapter 4](#).

JAZN APIs do not support user creation. Use either the Oracle Internet Directory APIs or Oracle Delegated Administration Services to create users.

Sample Code

The sample code that follows shows both standard J2EE and JAZN APIs being used to retrieve user information after authentication has occurred.

```
package oracle.security.jazn.samples.http;

import java.io.IOException;
import java.util.Date;
import java.util.Properties;
import javax.naming.*;
import javax.servlet.*;
import javax.servlet.http.*;
```



```

/**
 * A simple demo that exercises the Servlet security APIs.
 *
 */
public class CallerInfo extends HttpServlet {

    public CallerInfo()
    {
        super();
    }

    public void init(ServletConfig config)
        throws ServletException
    {
        super.init(config);
    }

    public void doGet(HttpServletRequest request, HttpServletResponse
        response)
        throws ServletException, IOException
    {
        ServletOutputStream out = response.getOutputStream();

        response.setContentType("text/html");
        out.println("<HTML><BODY bgcolor='#FFFFFF'>");

        //Standard J2EE APIs
        out.println("request.getRemoteUser = " +
            request.getRemoteUser() + "<br>");
        out.println("request.isUserInRole('FOO') = " +
            request.isUserInRole("FOO") + "<br>");
        out.println("request.isUserInRole('ar_manager') = " +
            request.isUserInRole("ar_manager") + "<br>");
        out.println("request.isUserInRole('ar_developer') = " +
            request.isUserInRole("ar_developer") + "<br>");
        out.println("request.getUserPrincipal = " +
            request.getUserPrincipal() + "<br>");

        //JAZN-LDAP APIs
        //Get the User principal from request
        com.evermind.security.User user =
(com.evermind.security.User)request.getUserPrincipal();
        //getDescription API Test
        try {
            java.lang.String s = user.getDescription();
            out.println("<b>getDescription</b> API Result: ["
                +s+ "]"<br>");
        }catch(Throwable e) {
            out.println("<b>getDescription</b> API FAILED: " +
                e.toString() + "<br>");
        }

        //getGroups API Test
        try {
            java.util.Set s = user.getGroups();
            out.println("<b>getGroups</b> API Result: [" +s+

```

```

        " ]<br>");
    }catch(Throwable e) {
        out.println("<b>getGroups</b> API FAILED: " +
            e.toString() + "<br>");
    }

    //getName API Test
    try {
        java.lang.String s = user.getName();
        out.println("<b>getName</b> API Result: [" +s+
            " ]<br>");
    }catch(Throwable e) {
        out.println("<b>getName</b> API FAILED: " +
            e.toString() + "<br>");
    }

    //hasPermission API Test
    try {
        com.evermind.server.rmi.RMIPermission p = new
            com.evermind.server.rmi.RMIPermission("login");
        boolean b = user.hasPermission(p);
        out.println("<b>hasPermission</b> API Result: [" + b
            + " ]<br>");
    }catch(Throwable e) {
        out.println("<b>hasPermission</b> API FAILED: " +
            e.toString() + "<br>");
    }

    //isMemberOf API Test
    try {
        java.util.Set s = user.getGroups();
        java.util.Iterator itr = s.iterator();
        boolean b = false;
        if(itr.hasNext())
        {
            b =
                user.isMemberOf((com.evermind.security.Group)itr.next());
        }
        out.println("<b>isMemberOf</b> API Result: [" +b+
            " ]<br>");
    }catch(Throwable e) {
        out.println("<b>isMemberOf</b> API FAILED: " +
            e.toString() + "<br>");
    }

    out.println("</BODY>");
    out.println("</HTML>");
}
}
}

```

JAAS Policy Management APIs

OC4J includes a highly scalable Java Authentication and Authorization Service (JAAS) provider, OracleAS JAAS Provider. J2EE applications integrated with Oracle Internet Directory can take advantage of the JAAS provider for enforcing fine-grained access control over protected resources.

OracleAS JAAS Provider supports using Oracle Internet Directory as the JAAS permissions and policies repository. OracleAS JAAS Provider is integrated with Oracle Internet Directory and OracleAS Single Sign-On to enhance application security.

This section includes the following topics

- [JAAS Policy Management](#)
- [Retrieving User Policies and Permissions using Standard JAAS APIs](#)

JAAS Policy Management

Permissions may be granted or revoked either by using the JAZN Admintool from the command line or programmatically, by using JAZN APIs.

The Admintool `jazn.jar` is found in the infrastructure installation under `$ORACLE_HOME/j2ee/home`. Set the `ORACLE_HOME` and `J2EE_HOME` environment variables before using it.

The following command line grants user `scott` permissions to read the file `foo.txt`. The realm name `scottsRealm` is defined in Oracle Internet Directory and the user name `scott` exists in Oracle Internet Directory:

```
java -jar jazn.jar -grantperm scottsRealm -user scott java.io.FilePermission
foo.txt, read
```

For more details on using the Admintool for User Management, see *Oracle Application Server Containers for J2EE Security Guide* Appendix B, "Using the JAZN Admintool".

To programmatically grant users permissions, you can use the JAZN's API as follows:

```
//get JAZNConfiguration related info
JAZNConfig jc = JAZNConfig.getJAZNConfig();

//create a Grantee for "scott"
RealmManager realmMgr = jc.getRealmManager();
Realm realm = realmMgr.getRealm("scottsRealm");
UserManager userMgr = realm.getUserManager();
final RealmUser user = userMgr.getUser("scott");

//grant scott file permission
JAZNPolicy policy = jc.getPolicy();

if ( policy != null) {
    Grantee gtee = new Grantee( (Principal) user);
    java.io.FilePermission fileperm = new java.io.FilePermission("foo.txt",
"read");
    policy.grant( gtee, fileperm);
}
```

For further details, see the *JAAS Provider API Reference* and the *Oracle Application Server Containers for J2EE Security Guide*.

Retrieving User Policies and Permissions using Standard JAAS APIs

Servlets may be run in either `doasprivileged` or `runasmode`. This causes them to be run in `Subject.doAsPrivileged` or `Subject.doAs` blocks, respectively. When servlets are run in either of these modes, you can check permissions by using either of two standard APIs: Policy APIs or AccessController. To retrieve policies, configure your servlet to use `doasprivileged` mode. For more information on how to

configure `doasprivileged` or `runas` mode, see "Configuring J2EE Authorization" in *Oracle Application Server Containers for J2EE Security Guide*.

The following code snippets show how to check permissions if user `scott` has permission to read `foo.txt`.

Checking or Listing Permissions Using `javax.security.auth.Policy`.

This approach allows you not only to check permissions, but also to list all the permissions granted to a user or group. If you only need to check the permissions granted to the user or group, and not code-based permissions, this approach is faster.

```
//create Permission
FilePermission perm = new FilePermission("/home/scott/foo.txt","read");
{
    javax.security.auth.Policy currPolicy =
        javax.security.auth.Policy.getPolicy();
    // Query policy now
    System.out.println("Policy permissions for this subject are " +
        currPolicy.getPermissions(Subject.getSubject(acc),null));

    //Check Permissions
    System.out.println("Policy.implies permission: "+ perm +" ? " +
        currPolicy.getPermissions(Subject.getSubject(acc),null).implies(perm));
}
```

Checking Permissions Using `AccessController`

Irrespective of whether the Security Manager is turned on or off, this code will check to see whether the subject or user executing this has permissions.

Note: If this snippet is executed in a servlet configured for `runas` mode, the code base also might require permission.

```
//create Permission
FilePermission perm = new FilePermission("/home/scott/foo.txt","read");
{
    //get current AccessControlContext
    AccessControlContext acc = AccessController.getContext();
    AccessController.checkPermission(perm);
}
```

For information about policy APIs provided by the OracleAS JAAS Provider, please see *Oracle Application Server Containers for J2EE Security Guide* Appendix A, "OracleAS JAAS Provider and Sample" and *Oracle Application Server Containers for J2EE Security Guide* Appendix B, "Using the JAZN Admintool"

For information about the Oracle Internet Directory Java APIs, see *Oracle Internet Directory API Reference* and [Chapter 4, "Using the Java API Extensions to JNDI"](#).

Part II

Oracle Internet Directory Programming Reference

Part II presents the standard APIs and the Oracle extensions to these APIs. It contains these chapters:

- [Chapter 11, "C API Reference"](#)
- [Chapter 12, "DBMS_LDAP PL/SQL Reference"](#)
- [Chapter 13, "Java API Reference"](#)
- [Chapter 14, "DBMS_LDAP_UTL PL/SQL Reference"](#)
- [Chapter 15, "DAS_URL Interface Reference"](#)
- [Chapter 16, "Centralized User Provisioning Java API Reference"](#)
- [Chapter 17, "Provisioning Integration PL/SQL API Reference"](#)

C API Reference

This chapter introduces the Oracle Internet Directory C API and provides examples of how to use it.

The chapter contains these topics:

- [About the Oracle Internet Directory C API](#)
- [Functions in the C API](#)
- [Sample C API Usage](#)
- [Required Header Files and Libraries for the C API](#)
- [Dependencies and Limitations of the C API](#)

About the Oracle Internet Directory C API

The Oracle Internet Directory SDK C API is based on LDAP Version 3 C API and Oracle extensions to support SSL.

You can use the Oracle Internet Directory API 10g Release 2 (10.1.2) in the following modes:

- **SSL**—All communication secured by using SSL
- **Non-SSL**—Client/server communication not secure

The API uses TCP/IP to connect to a directory server. When it does this, it uses, by default, an unencrypted channel. To use the SSL mode, you must use the Oracle SSL call interface. You determine which mode you are using by the presence or absence of the SSL calls in the API usage. You can easily switch between SSL and non-SSL modes.

See Also: "[Sample C API Usage](#)" on page 11-40 for more details on how to use the two modes

This section contains these topics:

- [Oracle Internet Directory SDK C API SSL Extensions](#)
- [The Functions at a Glance](#)

Oracle Internet Directory SDK C API SSL Extensions

Oracle SSL extensions to the LDAP API are based on standard SSL protocol. The SSL extensions provide encryption and decryption of data over the wire and authentication.

There are three modes of authentication:

- None—Neither client nor server is authenticated, and only SSL encryption is used
 - One-way—Only the server is authenticated by the client
 - Two-way—Both the server and the client are authenticated by each other
- The type of authentication is indicated by a parameter in the SSL interface call.

SSL Interface Calls

There is only one call required to enable SSL:

```
int ldap_init_SSL(Socketbuf *sb, text *sslwallet, text *sslwalletpasswd, int
sslauthmode)
```

The `ldap_init_SSL` call performs the necessary handshake between client and server using the standard SSL protocol. If the call is successful, then all subsequent communication happens over a secure connection.

Table 11–1 Arguments for SSL Interface Calls

Argument	Description
<code>sb</code>	Socket buffer handle returned by the <code>ldap_open</code> call as part of LDAP handle.
<code>sslwallet</code>	Location of the user wallet.
<code>sslwalletpasswd</code>	Password required to use the wallet.
<code>sslauthmode</code>	SSL authentication mode user wants to use. Possible values are: <ul style="list-style-type: none"> ■ <code>GSLC_SSL_NO_AUTH</code>—No authentication required ■ <code>GSLC_SSL_ONEWAY_AUTH</code>—Only server authentication required. ■ <code>GSLC_SSL_TWOWAY_AUTH</code>—Both server and client authentication required. <p>A return value of 0 indicates success. A nonzero return value indicates an error. The error code can be decoded by using the function <code>ldap_err2string</code>.</p>

See Also: ["Sample C API Usage"](#) on page 11-40

Wallet Support

depending on which authentication mode is being used, both the server and the client may require wallets to use the SSL feature. 10g Release 2 (10.1.2) of the API supports only the Oracle Wallet. You can create wallets by using Oracle Wallet Manager.

Functions in the C API

This section examines each of the functions and procedures in the C API. It explains their purpose and syntax. It also provides tips for using them.

The section contains the following topics:

- [The Functions at a Glance](#)
- [Initializing an LDAP Session](#)
- [LDAP Session Handle Options](#)
- [Authenticating to the Directory](#)

- [SASL Authentication Using Oracle Extensions](#)
- [SASL Authentication](#)
- [Working With Controls](#)
- [Closing the Session](#)
- [Performing LDAP Operations](#)
- [Abandoning an Operation](#)
- [Obtaining Results and Peeking Inside LDAP Messages](#)
- [Handling Errors and Parsing Results](#)
- [Stepping Through a List of Results](#)
- [Parsing Search Results](#)

The Functions at a Glance

Table 11-2 lists all of the functions and procedures in the C API and briefly explains their purpose.

Table 11-2 *Functions and Procedures in the C API*

Function or Procedure	Description
<code>ber_free</code>	Free the memory allocated for a BerElement structure
<code>ldap_abandon_ext</code> <code>ldap_abandon</code>	Cancel an asynchronous operation
<code>ldap_add_ext</code> <code>ldap_add_ext_s</code> <code>ldap_add</code> <code>ldap_add_s</code>	Add a new entry to the directory
<code>ldap_compare_ext</code> <code>ldap_compare_ext_s</code> <code>ldap_compare</code> <code>ldap_compare_s</code>	Compare entries in the directory
<code>ldap_count_entries</code>	Count the number of entries in a chain of search results
<code>ldap_count_values</code>	Count the string values of an attribute
<code>ldap_count_values_len</code>	Count the binary values of an attribute
<code>ora_ldap_create_clientctx</code>	Create a client context and returns a handle to it.
<code>ora_ldap_create_cred_hdl</code>	Create a credential handle.
<code>ldap_delete_ext</code> <code>ldap_delete_ext_s</code> <code>ldap_delete</code> <code>ldap_delete_s</code>	Delete an entry from the directory
<code>ora_ldap_destroy_clientctx</code>	Destroy the client context.
<code>ora_ldap_free_cred_hdl</code>	Destroy the credential handle.
<code>ldap_dn2ufn</code>	Converts the name into a more user friendly format
<code>ldap_err2string</code>	Get the error message for a specific error code
<code>ldap_explode_dn</code> <code>ldap_explode_rdn</code>	Split up a distinguished name into its components

Table 11-2 (Cont.) Functions and Procedures in the C API

Function or Procedure	Description
ldap_first_attribute	Get the name of the first attribute in an entry
ldap_first_entry	Get the first entry in a chain of search results
ora_ldap_get_cred_props	Retrieve properties associated with credential handle.
ldap_get_dn	Get the distinguished name for an entry
ldap_get_option	Access the current value of various session-wide parameters
ldap_get_values	Get the string values of an attribute
ldap_get_values_len	Get the binary values of an attribute
ldap_init ldap_open	Open a connection to an LDAP server
ora_ldap_init_SASL	Perform SASL authentication
ldap_memfree	Free memory allocated by an LDAP API function call
ldap_modify_ext ldap_modify_ext_s ldap_modify ldap_modify_s	Modify an entry in the directory
ldap_msgfree	Free the memory allocated for search results or other LDAP operation results
ldap_first_attribute ldap_next_attribute	Get the name of the next attribute in an entry
ldap_next_entry	Get the next entry in a chain of search results
ldap_perror (Deprecated)	Prints the message supplied in message.
ldap_rename ldap_rename_s	Modify the RDN of an entry in the directory
ldap_result2error (Deprecated)	Return the error code from result message.
ldap_result ldap_msgfree ldap_msgtype ldap_msgid	Check the results of an asynchronous operation
ldap_sasl_bind ldap_sasl_bind_s	General authentication to an LDAP server
ldap_search_ext ldap_search_ext_s ldap_search ldap_search_s	Search the directory
ldap_search_st	Search the directory with a timeout value
ldap_get_option ldap_set_option	Set the value of these parameters
ora_ldap_set_clientctx	Add properties to the client context handle.
ora_ldap_set_cred_props	Add properties to credential handle.

Table 11–2 (Cont.) Functions and Procedures in the C API

Function or Procedure	Description
ldap_simple_bind ldap_simple_bind_s ldap_sasl_bind ldap_sasl_bind_s	Simple authentication to an LDAP server
ldap_unbind_ext ldap_unbind ldap_unbind_s	End an LDAP session
ldap_value_free	Free the memory allocated for the string values of an attribute
ldap_value_free ldap_value_free_len	Free the memory allocated for the binary values of an attribute

This section lists all the calls available in the LDAP C API found in RFC 1823.

See Also: The following URL for a more detailed explanation of these calls:

<http://www.ietf.org>

Initializing an LDAP Session

The calls in this section initialize a session with an LDAP server.

ldap_init and ldap_open

ldap_init() initializes a session with an LDAP server, but does not open a connection. The server is not actually contacted until an operation is performed that requires it, allowing various options to be set after initialization. ldap_open() initializes a session and opens a connection. The two fulfill the same purpose and have the same syntax, but the first is preferred.

Syntax

```
LDAP *ldap_init
(
    const char    *hostname,
    int           portno
)
;
```

Parameters

Table 11–3 Parameters for Initializing an LDAP Session

Parameter	Description
hostname	Contains a space-separated list of host names or dotted strings representing the IP address of hosts running an LDAP server to connect to. Each host name in the list may include a port number. The two must be separated by a colon. The hosts are tried in the order listed until a successful connection occurs. Note: A suitable representation for including a literal IPv6[10] address in the host name parameter is desired, but has not yet been determined or implemented in practice.

Table 11–3 (Cont.) Parameters for Initializing an LDAP Session

Parameter	Description
portno	Contains the TCP port number to connect to. The default LDAP port of 389 can be obtained by supplying the constant <code>LDAP_PORT</code> . If <code>hostname</code> includes a port number, <code>portno</code> is ignored.

Usage Notes

`ldap_init()` and `ldap_open()` both return a session handle. This is a pointer to an opaque structure that must be passed to subsequent calls pertaining to the session. These routines return `NULL` if the session cannot be initialized. If the session cannot be initialized, check the error reporting mechanism for the operating system to see why the call failed.

Note that if you connect to an LDAPv2 server, one of the LDAP bind calls described later SHOULD be completed before other operations can be performed on the session. LDAPv3 does not require that a bind operation be completed before other operations are performed.

The calling program can set various attributes of the session by calling the routines described in the next section.

LDAP Session Handle Options

The LDAP session handle returned by `ldap_init()` is a pointer to an opaque data type representing an LDAP session. In RFC 1823 this data type was a structure exposed to the caller, and various fields in the structure could be set to control aspects of the session, such as size and time limits on searches.

In the interest of insulating callers from inevitable changes to this structure, these aspects of the session are now accessed through a pair of accessor functions, described in this section.

ldap_get_option and ldap_set_option

`ldap_get_option()` is used to access the current value of various session-wide parameters. `ldap_set_option()` is used to set the value of these parameters. Note that some options are read only and cannot be set; it is an error to call `ldap_set_option()` and attempt to set a read only option.

Note that if automatic referral following is enabled (the default), any connections created during the course of following referrals will inherit the options associated with the session that sent the original request that caused the referrals to be returned.

Syntax

```
int ldap_get_option
(
LDAP          *ld,
int           option,
void          *outvalue
)
;

int ldap_set_option
(
LDAP          *ld,
int           option,
const void    *invalue
)
```

```

)
;

#define LDAP_OPT_ON      ((void *)1)
#define LDAP_OPT_OFF    ((void *)0)

```

Parameters

Table 11-4 lists and describes the parameters for LDAP session handle options.

Table 11-4 Parameters for LDAP Session Handle Options

Parameters	Description
ld	The session handle. If this is NULL, a set of global defaults is accessed. New LDAP session handles created with ldap_init() or ldap_open() inherit their characteristics from these global defaults.
option	The name of the option being accessed or set. This parameter should be one of the constants listed and described in Table 11-5 on page 11-7. The hexadecimal value of the constant is listed in parentheses after the constant.
outvalue	The address of a place to put the value of the option. The actual type of this parameter depends on the setting of the option parameter. For outvalues of type char ** and LDAPControl **, a copy of the data that is associated with the LDAP session ld is returned. Callers should dispose of the memory by calling ldap_memfree() or ldap_controls_free(), depending on the type of data returned.
invalue	A pointer to the value the option is to be given. The actual type of this parameter depends on the setting of the option parameter. The data associated with invalue is copied by the API implementation to allow callers of the API to dispose of or otherwise change their copy of the data after a successful call to ldap_set_option(). If a value passed for invalue is invalid or cannot be accepted by the implementation, ldap_set_option() should return -1 to indicate an error.

Constants

Table 11-5 on page 11-7 lists and describes the constants for LDAP session handle options.

Table 11-5 Constants

Constant	Type for invalue parameter	Type for outvalue parameter	Description
LDAP_OPT_API_INFO(0x00)	Not applicable. Option is read only.	LDAPAPIInfo*	Used to retrieve some basic information about the LDAP API implementation at execution time. Applications need to be able to determine information about the particular API implementation they are using both at compile time and during execution. This option is read only and cannot be set.
ORA_LDAP_OPT_RFRL_CACHE	void* (LDAP_OPT_ON void* (LDAP_OPT_OFF)		This option determines whether referral cache is enabled or not. If this option is set to LDAP_OPT_ON, the cache is enabled; otherwise, the cache is disabled.
ORA_LDAP_OPT_RFRL_CACHE_SZ	int *	int *	This option sets the size of referral cache. The size is maximum size in terms of number of bytes the cache can grow to. It is set to 1MB by default.

Table 11–5 (Cont.) Constants

Constant	Type for invalue parameter	Type for outvalue parameter	Description
LDAP_OPT_DEREF(0x02)	int *	int *	Determines how aliases are handled during search. It should have one of the following values: LDAP_DEREF_NEVER (0x00), LDAP_DEREF_SEARCHING (0x01), LDAP_DEREF_FINDING (0x02), or LDAP_DEREF_ALWAYS (0x03). The LDAP_DEREF_SEARCHING value means aliases are dereferenced during the search but not when locating the base object of the search. The LDAP_DEREF_FINDING value means aliases are dereferenced when locating the base object but not during the search. The default value for this option is LDAP_DEREF_NEVER.
LDAP_OPT_SIZELIMIT(0x03)	int *	int *	A limit on the number of entries to return from a search. A value of LDAP_NO_LIMIT (0) means no limit. The default value for this option is LDAP_NO_LIMIT.
LDAP_OPT_TIMELIMIT(0x04)	int *	int *	A limit on the number of seconds to spend on a search. A value of LDAP_NO_LIMIT (0) means no limit. This value is passed to the server in the search request only; it does not affect how long the C LDAP API implementation itself will wait locally for search results. The timeout parameter passed to ldap_search_ext_s() or ldap_result()—both of which are described later in this document—can be used to specify both a local and server side time limit. The default value for this option is LDAP_NO_LIMIT.
LDAP_OPT_REFERRALS(0x08)	void *(LDAP_OPT_ON) void *(LDAP_OPT_OFF)	int *	Determines whether the LDAP library automatically follows referrals returned by LDAP servers or not. It may be set to one of the constants LDAP_OPT_ON or LDAP_OPT_OFF. Any non-null pointer value passed to ldap_set_option() enables this option. When the current setting is read using ldap_get_option(), a zero value means off and any nonzero value means on. By default, this option is turned on.
LDAP_OPT_RESTART(0x09)	void *(LDAP_OPT_ON) void *(LDAP_OPT_OFF)	int *	Determines whether LDAP input and output operations are automatically restarted if they stop prematurely. It may be set to either LDAP_OPT_ON or LDAP_OPT_OFF. Any non-null pointer value passed to ldap_set_option() enables this option. When the current setting is read using ldap_get_option(), a zero value means off and any nonzero value means on. This option is useful if an input or output operation can be interrupted prematurely—by a timer going off, for example. By default, this option is turned off.

Table 11–5 (Cont.) Constants

Constant	Type for invalue parameter	Type for outvalue parameter	Description
LDAP_OPT_PROTOCOL_VERSION(0x11)	int *	int *	This option indicates the version of the LDAP protocol used when communicating with the primary LDAP server. The option should be either LDAP_VERSION2 (2) or LDAP_VERSION3 (3). If no version is set, the default is LDAP_VERSION2 (2).
LDAP_OPT_SERVER_CONTROLS(0x12)	LDAPControl**	LDAPControl***	A default list of LDAP server controls to be sent with each request. See Also: "Working With Controls" on page 11-14
LDAP_OPT_CLIENT_CONTROLS(0x13)	LDAPControl**	LDAPControl***	A default list of client controls that affect the LDAP session. See Also: "Working With Controls" on page 11-14
LDAP_OPT_API_FEATURE_INFO(0x15)	Not applicable. Option is read only.	LDAPAPIFeatureInfo *	Used to retrieve version information about LDAP API extended features at execution time. Applications need to be able to determine information about the particular API implementation they are using both at compile time and during execution. This option is read only. It cannot be set.
LDAP_OPT_HOST_NAME(0x30)	char *	char **	The host name (or list of hosts) for the primary LDAP server. See the definition of the <code>hostname</code> parameter for <code>ldap_init()</code> to determine the syntax.
LDAP_OPT_ERROR_NUMBER(0x31)	int *	int *	The code of the most recent LDAP error during this session.
LDAP_OPT_ERROR_STRING(0x32)	char *	-	The message returned with the most recent LDAP error during this session.
LDAP_OPT_MATCHED_DN(0x33)	char *	char **	The matched DN value returned with the most recent LDAP error during this session.

Usage Notes

Both `ldap_get_option()` and `ldap_set_option()` return 0 if successful and -1 if an error occurs. If -1 is returned by either function, a specific error code may be retrieved by calling `ldap_get_option()` with an option value of `LDAP_OPT_ERROR_NUMBER`. Note that there is no way to retrieve a more specific error code if a call to `ldap_get_option()` with an option value of `LDAP_OPT_ERROR_NUMBER` fails.

When a call to `ldap_get_option()` succeeds, the API implementation **MUST NOT** change the state of the LDAP session handle or the state of the underlying implementation in a way that affects the behavior of future LDAP API calls. When a call to `ldap_get_option()` fails, the only session handle change permitted is setting the LDAP error code (as returned by the `LDAP_OPT_ERROR_NUMBER` option).

When a call to `ldap_set_option()` fails, it must not change the state of the LDAP session handle or the state of the underlying implementation in a way that affects the behavior of future LDAP API calls.

Standards track documents that extend this specification and specify new options should use values for option macros that are between 0x1000 and 0x3FFF inclusive. Private and experimental extensions should use values for the option macros that are between 0x4000 and 0x7FFF inclusive. All values less than 0x1000 and greater than 0x7FFF that are not defined in this document are reserved and should not be used. The following macro must be defined by C LDAP API implementations to aid extension implementers:

```
#define LDAP_OPT_PRIVATE_EXTENSION_BASE 0x4000 /* to 0x7FFF inclusive */
```

Authenticating to the Directory

The functions in this section are used to authenticate an LDAP client to an LDAP directory server.

ldap_sasl_bind, ldap_sasl_bind_s, ldap_simple_bind, and ldap_simple_bind_s

The `ldap_sasl_bind()` and `ldap_sasl_bind_s()` functions can be used to do general and extensible authentication over LDAP through the use of the Simple Authentication Security Layer. The routines both take the DN to bind as, the method to use, as a dotted-string representation of an object identifier (OID) identifying the method, and a `struct berval` holding the credentials. The special constant value `LDAP_SASL_SIMPLE` (`NULL`) can be passed to request simple authentication, or the simplified routines `ldap_simple_bind()` or `ldap_simple_bind_s()` can be used.

Syntax

```
int ldap_sasl_bind
(
    LDAP                *ld,
    const char          *dn,
    const char          *mechanism,
    const struct berval *cred,
    LDAPControl         **serverctrls,
    LDAPControl         **clientctrls,
    int                 *msgidp
);

int ldap_sasl_bind_s(
    LDAP                *ld,
    const char          *dn,
    const char          *mechanism,
    const struct berval *cred,
    LDAPControl         **serverctrls,
    LDAPControl         **clientctrls,
    struct berval       **servercredp
);

int ldap_simple_bind(
    LDAP                *ld,
    const char          *dn,
    const char          *passwd
);

int ldap_simple_bind_s(
    LDAP                *ld,
    const char          *dn,
```



```
const char          *passwd
);
```

The use of the following routines is deprecated and more complete descriptions can be found in RFC 1823:

- `int ldap_bind(LDAP *ld, const char *dn, const char *cred, int method);`
- `int ldap_bind_s(LDAP *ld, const char *dn, const char *cred, int method);`
- `int ldap_kerberos_bind(LDAP *ld, const char *dn);`
- `int ldap_kerberos_bind_s(LDAP *ld, const char *dn);`

Parameters

[Table 11-6](#) lists and describes the parameters for authenticating to the directory.

Table 11-6 *Parameters for Authenticating to the Directory*

Parameter	Description
<code>ld</code>	The session handle
<code>dn</code>	The name of the entry to bind as
<code>mechanism</code>	Either <code>LDAP_SASL_SIMPLE</code> (<code>NULL</code>) to get simple authentication, or a text string identifying the SASL method
<code>cred</code>	The credentials with which to authenticate. Arbitrary credentials can be passed using this parameter. The format and content of the credentials depends on the setting of the <code>mechanism</code> parameter.
<code>passwd</code>	For <code>ldap_simple_bind()</code> , the password to compare to the entry's <code>userPassword</code> attribute
<code>serverctrls</code>	List of LDAP server controls
<code>clientctrls</code>	List of client controls
<code>msgidp</code>	This result parameter will be set to the message id of the request if the <code>ldap_sasl_bind()</code> call succeeds
<code>servercredp</code>	This result parameter will be filled in with the credentials passed back by the server for mutual authentication, if given. An allocated <code>ber_val</code> structure is returned that should be disposed of by calling <code>ber_bvfree()</code> . <code>NULL</code> should be passed to ignore this field.

Usage Notes

Additional parameters for the deprecated routines are not described. Interested readers are referred to RFC 1823.

The `ldap_sasl_bind()` function initiates an asynchronous bind operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_sasl_bind()` places the message id of the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the result of the bind.

The `ldap_simple_bind()` function initiates a simple asynchronous bind operation and returns the message id of the operation initiated. A subsequent call to `ldap_result()`, described in, can be used to obtain the result of the bind. In case of error, `ldap_simple_bind()` will return `-1`, setting the session error parameters in the LDAP structure appropriately.

The synchronous `ldap_sasl_bind_s()` and `ldap_simple_bind_s()` functions both return the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not.

Note that if an LDAPv2 server is contacted, no other operations over the connection can be attempted before a bind call has successfully completed.

Subsequent bind calls can be used to re-authenticate over the same connection, and multistep SASL sequences can be accomplished through a sequence of calls to `ldap_sasl_bind()` or `ldap_sasl_bind_s()`.

See Also: ["Handling Errors and Parsing Results"](#) for more information about possible errors and how to interpret them

SASL Authentication Using Oracle Extensions

The function `ora_ldap_init_SASL()` can be used for SASL based authentication. It accepts these arguments:

- DN of the entity to be authenticated.
- SASL credential handle for the entity. (This handle can be managed using `ora_ldap_create_cred_hdl()`, `ora_ldap_set_cred_props()` and `ora_ldap_free_cred_hdl()` functions).
- SASL mechanism to be used.

This function encapsulates the SASL handshake between the client and the directory server for various standard SASL mechanisms thereby reducing the coding effort involved in establishing a SASL-based connection to the directory server.

Supported SASL mechanisms:

- DIGEST-MD5

The SASL API supports the authentication-only mode of DIGEST-MD5. The other two authentication modes addressing data privacy and data integrity are yet to be supported.

While authenticating against Oracle Internet Directory, the DN of the user has to be normalized before it is sent across to the server. This can be done either outside the SASL API using the `ora_ldap_normalize_dn()` function before the DN is passed on to the SASL API or with the SASL API by setting the `ORA_LDAP_CRED_SASL_NORM_AUTHDN` option in SASL credentials handle using `ora_ldap_set_cred_handle()`.

- EXTERNAL:

The SASL API and SASL implementation in Oracle Internet Directory use SSL authentication as one of the external authentication mechanisms.

Using this mechanism requires that the SSL connection (mutual authentication mode) be established to the directory server by using the `ora_ldap_init_SSL()` function. The `ora_ldap_init_SASL()` function can then be invoked with the `mechanism` argument as `EXTERNAL`. The directory server would then authenticate the user based on the user credentials in SSL connection.

`ora_ldap_create_cred_hdl`, `ora_ldap_set_cred_props`, `ora_ldap_get_cred_props`, and `ora_ldap_free_cred_hdl`

Use these functions to create and manage SASL credential handles. The `ora_ldap_create_cred_hdl` function should be used to create a SASL credential handle of

certain type based on the type of mechanism used for SASL authentication. The `ora_ldap_set_cred_props()` function can be used to add relevant credentials to the handle needed for SASL authentication. The `ora_ldap_get_cred_props()` function can be used for retrieving the properties stored in the credential handle, and the `ora_ldap_free_cred_hdl()` function should be used to destroy the handle after its use.

Syntax

```
OraLdapHandle ora_ldap_create_cred_hdl
(
    OraLdapClientCtx * clientCtx,
    int               credType
);

OraLdapHandle ora_ldap_set_cred_props
(
    OraLdapClientCtx * clientCtx,
    OraLdapHandle    cred,
    int               String[],
    void             * inProperty
);

OraLdapHandle ora_ldap_get_cred_props
(
    OraLdapClientCtx * clientCtx,
    OraLdapHandle    cred,
    int               String[],
    void             * outProperty
);

OraLdapHandle ora_ldap_free_cred_hdl
(
    OraLdapClientCtx * clientCtx,
    OraLdapHandle    cred
);
```

Parameters

Table 11-7 Parameters for Managing SASL Credentials

Parameter	Description
<code>clientCtx</code>	C API Client context. This can be managed using <code>ora_ldap_init_clientctx()</code> and <code>ora_ldap_free_clientctx()</code> functions.
<code>credType</code>	Type of credential handle specific to SASL mechanism.
<code>cred</code>	Credential handle containing SASL credentials needed for a specific SASL mechanism for SASL authentication.
<code>String[]</code>	Type of credential, which needs to be added to credential handle.
<code>inProperty</code>	One of the SASL Credentials to be stored in credential handle.
<code>outProperty</code>	One of the SASL credentials stored in credential handle.

SASL Authentication

`ora_ldap_init_SASL`, the lone function in this section, performs SASL authentication.

ora_ldap_init_SASL

ora_ldap_init_SASL performs authentication based on the mechanism specified as one of its input arguments.

Syntax

```
int ora_ldap_init_SASL
(
    OraLdapClientCtx * clientCtx,
    LDAP*ld,
    char* dn,
    char* mechanism,
    OraLdapHandle cred,
    LDAPControl**serverctrls,
    LDAPControl**clientctrls
);
```

Parameters

Table 11–8 Parameters for Managing SASL Credentials

Parameter	Description
clientCtx	C API Client context. This can be managed using ora_ldap_init_clientctx() and ora_ldap_free_clientctx() functions.
ld	Ldap session handle.
dn	User DN that requires authentication.
mechanism	SASL mechanism.
cred	Credentials needed for SASL authentication.
serverctrls	List of LDAP server controls
clientctrls	List of client controls

Working With Controls

LDAPv3 operations can be extended through the use of controls. Controls can be sent to a server or returned to the client with any LDAP message. These controls are referred to as server controls.

The LDAP API also supports a client-side extension mechanism through the use of client controls. These controls affect the behavior of the LDAP API only and are never sent to a server. A common data structure is used to represent both types of controls:

```
typedef struct ldapcontrol
{
    char          *ldctl_oid;
    struct berval ldctl_value;
    char          ldctl_iscritical;
} LDAPControl;
```

The fields in the ldapcontrol structure are described in [Table 11–9](#).

Table 11–9 Fields in ldapcontrol Structure

Field	Description
ldctl_oid	The control type, represented as a string.

Table 11–9 (Cont.) Fields in ldapcontrol Structure

Field	Description
ldctl_value	The data associated with the control (if any). To specify a zero-length value, set <code>ldctl_value.bv_len</code> to zero and <code>ldctl_value.bv_val</code> to a zero-length string. To indicate that no data is associated with the control, set <code>ldctl_value.bv_val</code> to NULL.
ldctl_iscritical	Indicates whether the control is critical or not. If this field is nonzero, the operation will only be carried out if the control is recognized by the server or the client. Note that the LDAP unbind and abandon operations have no server response. Clients should not mark server controls critical when used with these two operations.

Some LDAP API calls allocate an `ldapcontrol` structure or a NULL-terminated array of `ldapcontrol` structures. The following routines can be used to dispose of a single control or an array of controls:

```
void ldap_control_free( LDAPControl *ctrl );
void ldap_controls_free( LDAPControl **ctrls );
```

If the `ctrl` or `ctrls` parameter is NULL, these calls do nothing.

A set of controls that affect the entire session can be set using the `ldap_set_option()` function described in "[ldap_get_option and ldap_set_option](#)" on page 11-6. A list of controls can also be passed directly to some LDAP API calls such as `ldap_search_ext()`, in which case any controls set for the session through the use of `ldap_set_option()` are ignored. Control lists are represented as a NULL-terminated array of pointers to `ldapcontrol` structures.

Server controls are defined by LDAPv3 protocol extension documents; for example, a control has been proposed to support server-side sorting of search results.

One client control is defined in this document (described in the following section). Other client controls may be defined in future revisions of this document or in documents that extend this API.

Client-Controlled Referral Processing As described previously in "[LDAP Session Handle Options](#)" on page 11-6, applications can enable and disable automatic chasing of referrals on a session-wide basis by using the `ldap_set_option()` function with the `LDAP_OPT_REFERRALS` option. It is also useful to govern automatic referral chasing on per-request basis. A client control with an object identifier (OID) of `1.2.840.113556.1.4.616` exists to provide this functionality.

```
/* OID for referrals client control */
#define LDAP_CONTROL_REFERRALS          "1.2.840.113556.1.4.616"

/* Flags for referrals client control value */
#define LDAP_CHASE_SUBORDINATE_REFERRALS 0x00000020U
#define LDAP_CHASE_EXTERNAL_REFERRALS    0x00000040U
```

To create a referrals client control, the `ldctl_oid` field of an `LDAPControl` structure must be set to `LDAP_CONTROL_REFERRALS` (`"1.2.840.113556.1.4.616"`) and the `ldctl_value` field must be set to a four-octet value that contains a set of flags. The `ldctl_value.bv_len` field must always be set to 4. The `ldctl_value.bv_val` field must point to a four-octet integer flags value. This flags value can be set to zero to disable automatic chasing of referrals and LDAPv3 references altogether. Alternatively, the flags value can be set to the value `LDAP_CHASE_SUBORDINATE_REFERRALS` (`0x00000020U`) to indicate that only LDAPv3 search continuation

references are to be automatically chased by the API implementation, to the value `LDAP_CHASE_EXTERNAL_REFERRALS` (0x00000040U) to indicate that only LDAPv3 referrals are to be automatically chased, or the logical OR of the two flag values (0x00000060U) to indicate that both referrals and references are to be automatically chased.

See Also: "Directory Schema Administration" in *Oracle Internet Directory Administrator's Guide* for more information about object identifiers

Closing the Session

Use the functions in this section to unbind from the directory, to close open connections, and to dispose of the session handle.

`ldap_unbind`, `ldap_unbind_ext`, and `ldap_unbind_s`

`ldap_unbind_ext()`, `ldap_unbind()`, and `ldap_unbind_s()` all work synchronously in the sense that they send an unbind request to the server, close all open connections associated with the LDAP session handle, and dispose of all resources associated with the session handle before returning. Note, however, that there is no server response to an LDAP unbind operation. All three of the unbind functions return `LDAP_SUCCESS` (or another LDAP error code if the request cannot be sent to the LDAP server). After a call to one of the unbind functions, the session handle `ld` is invalid and it is illegal to make any further LDAP API calls using `ld`.

The `ldap_unbind()` and `ldap_unbind_s()` functions behave identically. The `ldap_unbind_ext()` function allows server and client controls to be included explicitly, but note that since there is no server response to an unbind request there is no way to receive a response to a server control sent with an unbind request.

Syntax

```
int ldap_unbind_ext( LDAP *ld, LDAPControl **serverctrls,
LDAPControl **clientctrls );
int ldap_unbind( LDAP *ld );
int ldap_unbind_s( LDAP *ld );
```

Parameters

Table 11–10 Parameters for Closing the Session

Parameter	Description
<code>ld</code>	The session handle
<code>serverctrls</code>	List of LDAP server controls
<code>clientctrls</code>	List of client controls
<code>clientctrls</code>	

Performing LDAP Operations

Use the functions in this section to search the LDAP directory and to return a requested set of attributes for each entry matched.

ldap_search_ext, ldap_search_ext_s, ldap_search, and ldap_search_s

The `ldap_search_ext()` function initiates an asynchronous search operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_search_ext()` places the message id of the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the results from the search. These results can be parsed using the result parsing routines described in detail later.

Similar to `ldap_search_ext()`, the `ldap_search()` function initiates an asynchronous search operation and returns the message id of the operation initiated. As for `ldap_search_ext()`, a subsequent call to `ldap_result()` can be used to obtain the result of the bind. In case of error, `ldap_search()` will return `-1`, setting the session error parameters in the LDAP structure appropriately.

The synchronous `ldap_search_ext_s()`, `ldap_search_s()`, and `ldap_search_st()` functions all return the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not. Entries returned from the search, if any, are contained in the `res` parameter. This parameter is opaque to the caller. Entries, attributes, values, and so on, can be extracted by calling the parsing routines described in this section. The results contained in `res` should be freed when no longer in use by calling `ldap_msgfree()`, which is described later.

The `ldap_search_ext()` and `ldap_search_ext_s()` functions support LDAPv3 server controls, client controls, and allow varying size and time limits to be easily specified for each search operation. The `ldap_search_st()` function is identical to `ldap_search_s()` except that it takes an additional parameter specifying a local timeout for the search. The local search timeout is used to limit the amount of time the API implementation will wait for a search to complete. After the local search timeout expires, the API implementation will send an abandon operation to stop the search operation.

See Also: ["Handling Errors and Parsing Results"](#) for more information about possible errors and how to interpret them

Syntax

```
int ldap_search_ext
(
    LDAP          *ld,
    const char    *base,
    int           scope,
    const char    *filter,
    char          **attrs,
    int           attronly,
    LDAPControl  **serverctrls,
    LDAPControl  **clientctrls,
    struct timeval *timeout,
    int           sizelimit,
    int           *msgidp
);

int ldap_search_ext_s
(
    LDAP          *ld,
    const char    *base,
    int           scope,
    const char    *filter,
    char          **attrs,
```

```

int          attronly,
LDAPControl **serverctrls,
LDAPControl **clientctrls,
struct timeval *timeout,
int          sizelimit,
LDAPMessage **res
);

int ldap_search
(
LDAP          *ld,
const char    *base,
int           scope,
const char    *filter,
char          **attrs,
int           attronly
);

int ldap_search_s
(
LDAP          *ld,
const char    *base,
int           scope,
const char    *filter,
char          **attrs,
int           attronly,
LDAPMessage  **res
);

int ldap_search_st
);

LDAP          *ld,
const char    *base,
int           scope,
const char    *filter,
char          **attrs,
int           attronly,
struct timeval *timeout,
LDAPMessage  **res
);

```

Parameters

Table 11-11 lists and describes the parameters for search operations.

Table 11-11 Parameters for Search Operations

Parameter	Description
ld	The session handle.
base	The DN of the entry at which to start the search.
scope	One of LDAP_SCOPE_BASE (0x00), LDAP_SCOPE_ONELEVEL (0x01), or LDAP_SCOPE_SUBTREE (0x02), indicating the scope of the search.
filter	A character string representing the search filter. The value NULL can be passed to indicate that the filter "(objectclass=*)" which matches all entries is to be used. Note that if the caller of the API is using LDAPv2, only a subset of the filter functionality can be successfully used.

Table 11–11 (Cont.) Parameters for Search Operations

Parameter	Description
<code>attrs</code>	A NULL-terminated array of strings indicating which attributes to return for each matching entry. Passing NULL for this parameter causes all available user attributes to be retrieved. The special constant string <code>LDAP_NO_ATTRS ("1.1")</code> may be used as the only string in the array to indicate that no attribute types are to be returned by the server. The special constant string <code>LDAP_ALL_USER_ATTRS ("*")</code> can be used in the <code>attrs</code> array along with the names of some operational attributes to indicate that all user attributes plus the listed operational attributes are to be returned.
<code>attrsonly</code>	A boolean value that must be zero if both attribute types and values are to be returned, and nonzero if only types are wanted.
<code>timeout</code>	For the <code>ldap_search_st()</code> function, this specifies the local search timeout value (if it is NULL, the timeout is infinite). If a zero timeout (where <code>tv_sec</code> and <code>tv_usec</code> are both zero) is passed, API implementations should return <code>LDAP_PARAM_ERROR</code> . For the <code>ldap_search_ext()</code> and <code>ldap_search_ext_s()</code> functions, the timeout parameter specifies both the local search timeout value and the operation time limit that is sent to the server within the search request. Passing a NULL value for timeout causes the global default timeout stored in the LDAP session handle (set by using <code>ldap_set_option()</code> with the <code>LDAP_OPT_TIMELIMIT</code> parameter) to be sent to the server with the request but an infinite local search timeout to be used. If a zero timeout (where <code>tv_sec</code> and <code>tv_usec</code> are both zero) is passed in, API implementations should return <code>LDAP_PARAM_ERROR</code> . If a zero value for <code>tv_sec</code> is used but <code>tv_usec</code> is nonzero, an operation time limit of 1 should be passed to the LDAP server as the operation time limit. For other values of <code>tv_sec</code> , the <code>tv_sec</code> value itself should be passed to the LDAP server.
<code>sizelimit</code>	For the <code>ldap_search_ext()</code> and <code>ldap_search_ext_s()</code> calls, this is a limit on the number of entries to return from the search. A value of <code>LDAP_NO_LIMIT (0)</code> means no limit.
<code>res</code>	For the synchronous calls, this is a result parameter which will contain the results of the search upon completion of the call. If no results are returned, <code>*res</code> is set to NULL.
<code>serverctrls</code>	List of LDAP server controls.
<code>clientctrls</code>	List of client controls.
<code>msgidp</code>	This result parameter will be set to the message id of the request if the <code>ldap_search_ext()</code> call succeeds. There are three options in the session handle <code>ld</code> which potentially affect how the search is performed. They are: <ul style="list-style-type: none"> ▪ <code>LDAP_OPT_SIZELIMIT</code>—A limit on the number of entries to return from the search. A value of <code>LDAP_NO_LIMIT (0)</code> means no limit. Note that the value from the session handle is ignored when using the <code>ldap_search_ext()</code> or <code>ldap_search_ext_s()</code> functions. ▪ <code>LDAP_OPT_TIMELIMIT</code>—A limit on the number of seconds to spend on the search. A value of <code>LDAP_NO_LIMIT (0)</code> means no limit. Note that the value from the session handle is ignored when using the <code>ldap_search_ext()</code> or <code>ldap_search_ext_s()</code> functions. ▪ <code>LDAP_OPT_DEREF</code>—One of <code>LDAP_DEREF_NEVER (0x00)</code>, <code>LDAP_DEREF_SEARCHING (0x01)</code>, <code>LDAP_DEREF_FINDING (0x02)</code>, or <code>LDAP_DEREF_ALWAYS (0x03)</code>, specifying how aliases are handled during the search. The <code>LDAP_DEREF_SEARCHING</code> value means aliases are dereferenced during the search but not when locating the base object of the search. The <code>LDAP_DEREF_FINDING</code> value means aliases are dereferenced when locating the base object but not during the search.

Reading an Entry

LDAP does not support a read operation directly. Instead, this operation is emulated by a search with base set to the DN of the entry to read, scope set to `LDAP_SCOPE_`

BASE, and filter set to "(objectclass=*)" or NULL. The `attrs` parameter contains the list of attributes to return.

Listing the Children of an Entry

LDAP does not support a list operation directly. Instead, this operation is emulated by a search with base set to the DN of the entry to list, scope set to `LDAP_SCOPE_ONELEVEL`, and filter set to "(objectclass=*)" or NULL. The parameter `attrs` contains the list of attributes to return for each child entry.

`ldap_compare_ext`, `ldap_compare_ext_s`, `ldap_compare`, and `ldap_compare_s`

Use these routines to compare an attribute value assertion against an LDAP entry.

The `ldap_compare_ext()` function initiates an asynchronous compare operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_compare_ext()` places the message id of the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the result of the compare.

Similar to `ldap_compare_ext()`, the `ldap_compare()` function initiates an asynchronous compare operation and returns the message id of the operation initiated. As for `ldap_compare_ext()`, a subsequent call to `ldap_result()` can be used to obtain the result of the bind. In case of error, `ldap_compare()` will return `-1`, setting the session error parameters in the LDAP structure appropriately.

The synchronous `ldap_compare_ext_s()` and `ldap_compare_s()` functions both return the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not.

The `ldap_compare_ext()` and `ldap_compare_ext_s()` functions support LDAPv3 server controls and client controls.

See Also: ["Handling Errors and Parsing Results"](#) for more information about possible errors and how to interpret them

Syntax

```
int ldap_compare_ext
(
    LDAP          *ld,
    const char    *dn,
    const char    *attr,
    const struct berval *bvalue,
    LDAPControl  **serverctrls,
    LDAPControl  **clientctrls,
    int           *msgidp
);

int ldap_compare_ext_s
(
    LDAP          *ld,
    const char    *dn,
    const char    *attr,
    const struct berval *bvalue,
    LDAPControl  **serverctrls,
    LDAPControl  **clientctrls
);

int ldap_compare
(
```

```

LDAP          *ld,
const char    *dn,
const char    *attr,
const char    *value
);
int ldap_compare_s
(
LDAP          *ld,
const char    *dn,
const char    *attr,
const char    *value
);

```

Parameters

[Table 11–12](#) lists and describes the parameters for compare operations.

Table 11–12 Parameters for Compare Operations

Parameter	Description
ld	The session handle.
dn	The name of the entry to compare against.
attr	The attribute to compare against.
bvalue	The attribute value to compare against those found in the given entry. This parameter is used in the extended routines and is a pointer to a <code>struct berval</code> so it is possible to compare binary values.
value	A string attribute value to compare against, used by the <code>ldap_compare()</code> and <code>ldap_compare_s()</code> functions. Use <code>ldap_compare_ext()</code> or <code>ldap_compare_ext_s()</code> if you need to compare binary values.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.
msgidp	This result parameter will be set to the message id of the request if the <code>ldap_compare_ext()</code> call succeeds.

`ldap_modify_ext`, `ldap_modify_ext_s`, `ldap_modify`, and `ldap_modify_s`

Use these routines to modify an existing LDAP entry.

The `ldap_modify_ext()` function initiates an asynchronous modify operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_modify_ext()` places the message id of the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the result of the modify.

Similar to `ldap_modify_ext()`, the `ldap_modify()` function initiates an asynchronous modify operation and returns the message id of the operation initiated. As for `ldap_modify_ext()`, a subsequent call to `ldap_result()` can be used to obtain the result of the modify. In case of error, `ldap_modify()` will return `-1`, setting the session error parameters in the LDAP structure appropriately.

The synchronous `ldap_modify_ext_s()` and `ldap_modify_s()` functions both return the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not.

The `ldap_modify_ext()` and `ldap_modify_ext_s()` functions support LDAPv3 server controls and client controls.

See Also: ["Handling Errors and Parsing Results"](#) for more information about possible errors and how to interpret them

Syntax

```
typedef struct ldapmod
{
int          mod_op;
char         *mod_type;
union mod_vals_u
{
char         **modv_strvals;
struct berval **modv_bvals;
} mod_vals;
} LDAPMod;
#define mod_values      mod_vals.modv_strvals
#define mod_bvalues    mod_vals.modv_bvals

int ldap_modify_ext
(
LDAP          *ld,
const char    *dn,
LDAPMod       **mods,
LDAPControl   **serverctrls,
LDAPControl   **clientctrls,
int           *msgidp
);

int ldap_modify_ext_s
(
LDAP          *ld,
const char    *dn,
LDAPMod       **mods,
LDAPControl   **serverctrls,
LDAPControl   **clientctrls
);

int ldap_modify
(
LDAP          *ld,
const char    *dn,
LDAPMod       **mods
);

int ldap_modify_s
(
LDAP          *ld,
const char    *dn,
LDAPMod       **mods
);
```

Parameters

[Table 11-13](#) lists and describes the parameters for modify operations.

Table 11-13 Parameters for Modify Operations

Parameter	Description
ld	The session handle
dn	The name of the entry to modify

Table 11–13 (Cont.) Parameters for Modify Operations

Parameter	Description
<code>mods</code>	A NULL-terminated array of modifications to make to the entry
<code>serverctrls</code>	List of LDAP server controls
<code>clientctrls</code>	List of client controls
<code>msgidp</code>	This result parameter will be set to the message id of the request if the <code>ldap_modify_ext()</code> call succeeds

Table 11–14 lists and describes the fields in the `LDAPMod` structure.

Table 11–14 Fields in LDAPMod Structure

Field	Description
<code>mod_op</code>	The modification operation to perform. It must be one of <code>LDAP_MOD_ADD</code> (0x00), <code>LDAP_MOD_DELETE</code> (0x01), or <code>LDAP_MOD_REPLACE</code> (0x02). This field also indicates the type of values included in the <code>mod_vals</code> union. It is logically ORed with <code>LDAP_MOD_BVALUES</code> (0x80) to select the <code>mod_bvalues</code> form. Otherwise, the <code>mod_values</code> form is used.
<code>mod_type</code>	The type of the attribute to modify.
<code>mod_vals</code>	The values (if any) to add, delete, or replace. Only one of the <code>mod_values</code> or <code>mod_bvalues</code> variants can be used, selected by ORing the <code>mod_op</code> field with the constant <code>LDAP_MOD_BVALUES</code> . <code>mod_values</code> is a NULL-terminated array of zero-terminated strings and <code>mod_bvalues</code> is a NULL-terminated array of <code>berval</code> structures that can be used to pass binary values such as images.

Usage Notes

For `LDAP_MOD_ADD` modifications, the given values are added to the entry, creating the attribute if necessary.

For `LDAP_MOD_DELETE` modifications, the given values are deleted from the entry, removing the attribute if no values remain. If the entire attribute is to be deleted, the `mod_vals` field can be set to `NULL`.

For `LDAP_MOD_REPLACE` modifications, the attribute will have the listed values after the modification, having been created if necessary, or removed if the `mod_vals` field is `NULL`. All modifications are performed in the order in which they are listed.

`ldap_rename` and `ldap_rename_s`

Use these routines to change the name of an entry.

The `ldap_rename()` function initiates an asynchronous modify DN operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_rename()` places the DN message id of the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the result of the rename.

The synchronous `ldap_rename_s()` returns the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not.

The `ldap_rename()` and `ldap_rename_s()` functions both support LDAPv3 server controls and client controls.

See Also: ["Handling Errors and Parsing Results"](#) for more information about possible errors and how to interpret them

Syntax

```
int ldap_rename
(
LDAP          *ld,
const char    *dn,
const char    *newrdn,
const char    *newparent,
int           deleteoldrdn,
LDAPControl   **serverctrls,
LDAPControl   **clientctrls,
int           *msgidp
);
```

```
int ldap_rename_s
(
LDAP          *ld,
const char    *dn,
const char    *newrdn,
const char    *newparent,
int           deleteoldrdn,
LDAPControl   **serverctrls,
LDAPControl   **clientctrls
);
```

The use of the following routines is deprecated and more complete descriptions can be found in RFC 1823:

```
int ldap_modrdn
(
LDAP          *ld,
const char    *dn,
const char    *newrdn
);
```

```
int ldap_modrdn_s
(
LDAP          *ld,
const char    *dn,
const char    *newrdn
);
```

```
int ldap_modrdn2
(
LDAP          *ld,
const char    *dn,
const char    *newrdn,
int           deleteoldrdn
);
```

```
int ldap_modrdn2_s
(
LDAP          *ld,
const char    *dn,
const char    *newrdn,
int           deleteoldrdn
);
```

Parameters

[Table 11–15](#) lists and describes the parameters for rename operations.

Table 11–15 Parameters for Rename Operations

Parameter	Description
ld	The session handle.
dn	The name of the entry whose DN is to be changed.
newrdn	The new RDN to give the entry.
newparent	The new parent, or superior entry. If this parameter is <code>NULL</code> , only the RDN of the entry is changed. The root DN should be specified by passing a zero length string, <code>" "</code> . The <code>newparent</code> parameter should always be <code>NULL</code> when using version 2 of the LDAP protocol; otherwise the server's behavior is undefined.
deleteoldrdn	This parameter only has meaning on the rename routines if <code>newrdn</code> is different than the old RDN. It is a boolean value, if nonzero indicating that the old RDN value is to be removed, if zero indicating that the old RDN value is to be retained as non-distinguished values of the entry.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.
msgidp	This result parameter will be set to the message id of the request if the <code>ldap_rename()</code> call succeeds.

ldap_add_ext, ldap_add_ext_s, ldap_add, and ldap_add_s

Use these functions to add entries to the LDAP directory.

The `ldap_add_ext()` function initiates an asynchronous add operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_add_ext()` places the message id of the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the result of the add.

Similar to `ldap_add_ext()`, the `ldap_add()` function initiates an asynchronous add operation and returns the message id of the operation initiated. As for `ldap_add_ext()`, a subsequent call to `ldap_result()` can be used to obtain the result of the add. In case of error, `ldap_add()` will return `-1`, setting the session error parameters in the LDAP structure appropriately.

The synchronous `ldap_add_ext_s()` and `ldap_add_s()` functions both return the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not.

The `ldap_add_ext()` and `ldap_add_ext_s()` functions support LDAPv3 server controls and client controls.

See Also: ["Handling Errors and Parsing Results"](#) for more information about possible errors and how to interpret them

Syntax

```
int ldap_add_ext
(
    LDAP          *ld,
    const char    *dn,
    LDAPMod       **attrs,
    LDAPControl   **serverctrls,
```

```

LDAPControl    **clientctrls,
int            *msgidp
);

int ldap_add_ext_s
(
LDAP          *ld,
const char   *dn,
LDAPMod      **attrs,
LDAPControl  **serverctrls,
LDAPControl  **clientctrls
);

int ldap_add
(
LDAP          *ld,
const char   *dn,
LDAPMod      **attrs
);

int ldap_add_s
(
LDAP          *ld,
const char   *dn,
LDAPMod      **attrs
);

```

Parameters

[Table 11–16](#) lists and describes the parameters for add operations.

Table 11–16 Parameters for Add Operations

Parameter	Description
ld	The session handle.
dn	The name of the entry to add.
attrs	The entry attributes, specified using the LDAPMod structure defined for ldap_modify(). The mod_type and mod_vals fields must be filled in. The mod_op field is ignored unless ORED with the constant LDAP_MOD_BVALUES, used to select the mod_bvalues case of the mod_vals union.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.
msgidp	This result parameter will be set to the message id of the request if the ldap_add_ext() call succeeds.

Usage Notes

Note that the parent of the entry being added must already exist or the parent must be empty—that is, equal to the root DN—for an add to succeed.

ldap_delete_ext, ldap_delete_ext_s, ldap_delete, and ldap_delete_s

Use these functions to delete a leaf entry from the LDAP directory.

The ldap_delete_ext() function initiates an asynchronous delete operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. If successful, ldap_delete_ext() places the message id of

the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the result of the delete.

Similar to `ldap_delete_ext()`, the `ldap_delete()` function initiates an asynchronous delete operation and returns the message id of the operation initiated. As for `ldap_delete_ext()`, a subsequent call to `ldap_result()` can be used to obtain the result of the delete. In case of error, `ldap_delete()` will return `-1`, setting the session error parameters in the LDAP structure appropriately.

The synchronous `ldap_delete_ext_s()` and `ldap_delete_s()` functions both return the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not.

The `ldap_delete_ext()` and `ldap_delete_ext_s()` functions support LDAPv3 server controls and client controls.

See Also: ["Handling Errors and Parsing Results"](#) for more information about possible errors and how to interpret them

Syntax

```
int ldap_delete_ext
(
    LDAP          *ld,
    const char    *dn,
    LDAPControl   **serverctrls,
    LDAPControl   **clientctrls,
    int           *msgidp
);
```

```
int ldap_delete_ext_s
(
    LDAP          *ld,
    const char    *dn,
    LDAPControl   **serverctrls,
    LDAPControl   **clientctrls
);
```

int ldap_delete

```
(
    LDAP          *ld,
    const char    *dn
);
```

```
int ldap_delete_s
(
    LDAP          *ld,
    const char    *dn
);
```

Parameters

[Table 11-17](#) lists and describes the parameters for delete operations.

Table 11-17 Parameters for Delete Operations

Parameter	Description
<code>ld</code>	The session handle.
<code>dn</code>	The name of the entry to delete.

Table 11–17 (Cont.) Parameters for Delete Operations

Parameter	Description
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.
msgidp	This result parameter will be set to the message id of the request if the <code>ldap_delete_ext()</code> call succeeds.

Usage Notes

Note that the entry to delete must be a leaf entry—that is, it must have no children. Deletion of entire subtrees in a single operation is not supported by LDAP.

ldap_extended_operation and ldap_extended_operation_s

These routines enable extended LDAP operations to be passed to the server, providing a general protocol extensibility mechanism.

The `ldap_extended_operation()` function initiates an asynchronous extended operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_extended_operation()` places the message id of the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the result of the extended operation which can be passed to `ldap_parse_extended_result()` to obtain the object identifier (OID) and data contained in the response.

The synchronous `ldap_extended_operation_s()` function returns the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not. The `retoid` and `retdata` parameters are filled in with the OID and data from the response. If no OID or data was returned, these parameters are set to `NULL`.

The `ldap_extended_operation()` and `ldap_extended_operation_s()` functions both support LDAPv3 server controls and client controls.

See Also: ["Handling Errors and Parsing Results"](#) for more information about possible errors and how to interpret them

Syntax

```
int ldap_extended_operation
(
    LDAP                *ld,
    const char          *requestoid,
    const struct berval *requestdata,
    LDAPControl         **serverctrls,
    LDAPControl         **clientctrls,
    int                 *msgidp
);

int ldap_extended_operation_s
(
    LDAP                *ld,
    const char          *requestoid,
    const struct berval *requestdata,
    LDAPControl         **serverctrls,
    LDAPControl         **clientctrls,
    char                **retoidp,
    struct berval       **retdatap
);
```

```
);
```

Parameters

[Table 11–18](#) lists and describes the parameters for extended operations.

Table 11–18 *Parameters for Extended Operations*

Parameter	Description
ld	The session handle
requestoid	The dotted-OID text string naming the request
requestdata	The arbitrary data needed by the operation (if <code>NULL</code> , no data is sent to the server)
serverctrls	List of LDAP server controls
clientctrls	List of client controls
msgidp	This result parameter will be set to the message id of the request if the <code>ldap_extended_operation()</code> call succeeds.
retoidp	Pointer to a character string that will be set to an allocated, dotted-OID text string returned by the server. This string should be disposed of using the <code>ldap_memfree()</code> function. If no OID was returned, <code>*retoidp</code> is set to <code>NULL</code> .
retdatap	Pointer to a <code>berval</code> structure pointer that will be set an allocated copy of the data returned by the server. This <code>struct berval</code> should be disposed of using <code>ber_bvfree()</code> . If no data is returned, <code>*retdatap</code> is set to <code>NULL</code> .

Abandoning an Operation

Use the functions in this section to abandon an operation in progress:

`ldap_abandon_ext` and `ldap_abandon`

`ldap_abandon_ext()` abandons the operation with message id `msgid` and returns the constant `LDAP_SUCCESS` if the abandon was successful or another LDAP error code if not.

`ldap_abandon()` is identical to `ldap_abandon_ext()` except that it does not accept client or server controls and it returns zero if the abandon was successful, `-1` otherwise.

After a successful call to `ldap_abandon()` or `ldap_abandon_ext()`, results with the given message id are never returned from a subsequent call to `ldap_result()`. There is no server response to LDAP abandon operations.

Syntax

```
int ldap_abandon_ext
(
    LDAP          *ld,
    int           msgid,
    LDAPControl   **serverctrls,
    LDAPControl   **clientctrls
);

int ldap_abandon
(
    LDAP          *ld,
    int           msgid
);
```

Parameters

[Table 11–19](#) lists and describes the parameters for abandoning an operation.

Table 11–19 *Parameters for Abandoning an Operation*

Parameter	Description
ld	The session handle.
msgid	The message id of the request to be abandoned.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.

See Also: ["Handling Errors and Parsing Results"](#) for more information about possible errors and how to interpret them

Obtaining Results and Peeking Inside LDAP Messages

Use the functions in this section to return the result of an operation initiated asynchronously. They identify messages by type and by ID.

ldap_result, ldap_msgtype, and ldap_msgid

`ldap_result()` is used to obtain the result of a previous asynchronously initiated operation. Note that depending on how it is called, `ldap_result()` can actually return a list or "chain" of result messages. The `ldap_result()` function only returns messages for a single request, so for all LDAP operations other than search only one result message is expected; that is, the only time the "result chain" can contain more than one message is if results from a search operation are returned.

Once a chain of messages has been returned to the caller, it is no longer tied in any caller-visible way to the LDAP request that produced it. Therefore, a chain of messages returned by calling `ldap_result()` or by calling a synchronous search routine will never be affected by subsequent LDAP API calls (except for `ldap_msgfree()` which is used to dispose of a chain of messages).

`ldap_msgfree()` frees the result messages (possibly an entire chain of messages) obtained from a previous call to `ldap_result()` or from a call to a synchronous search routine.

`ldap_msgtype()` returns the type of an LDAP message. `ldap_msgid()` returns the message ID of an LDAP message.

Syntax

```
int ldap_result
(
LDAP          *ld,
int           msgid,
int           all,
struct timeval *timeout,
LDAPMessage  **res
);
int ldap_msgfree( LDAPMessage *res );
int ldap_msgtype( LDAPMessage *res );
int ldap_msgid( LDAPMessage *res );
```

Parameters

[Table 11–20](#) on page 11-31 lists and describes the parameters for obtaining results and peeling inside LDAP messages.

Table 11–20 Parameters for Obtaining Results and Peeking Inside LDAP Messages

Parameter	Description
ld	The session handle.
msgid	The message id of the operation whose results are to be returned, the constant <code>LDAP_RES_UNSOLICITED</code> (0) if an unsolicited result is desired, or the constant <code>LDAP_RES_ANY</code> (-1) if any result is desired.
all	Specifies how many messages will be retrieved in a single call to <code>ldap_result()</code> . This parameter only has meaning for search results. Pass the constant <code>LDAP_MSG_ONE</code> (0x00) to retrieve one message at a time. Pass <code>LDAP_MSG_ALL</code> (0x01) to request that all results of a search be received before returning all results in a single chain. Pass <code>LDAP_MSG_RECEIVED</code> (0x02) to indicate that all messages retrieved so far are to be returned in the result chain.
timeout	A timeout specifying how long to wait for results to be returned. A NULL value causes <code>ldap_result()</code> to block until results are available. A timeout value of zero seconds specifies a polling behavior.
res	For <code>ldap_result()</code> , a result parameter that will contain the result of the operation. If no results are returned, <code>*res</code> is set to NULL. For <code>ldap_msgfree()</code> , the result chain to be freed, obtained from a previous call to <code>ldap_result()</code> , <code>ldap_search_s()</code> , or <code>ldap_search_st()</code> . If <code>res</code> is NULL, nothing is done and <code>ldap_msgfree()</code> returns zero.

Usage Notes

Upon successful completion, `ldap_result()` returns the type of the first result returned in the `res` parameter. This will be one of the following constants.

`LDAP_RES_BIND` (0x61)

`LDAP_RES_SEARCH_ENTRY` (0x64)

`LDAP_RES_SEARCH_REFERENCE` (0x73) -- new in LDAPv3

`LDAP_RES_SEARCH_RESULT` (0x65)

`LDAP_RES_MODIFY` (0x67)

`LDAP_RES_ADD` (0x69)

`LDAP_RES_DELETE` (0x6B)

`LDAP_RES_MOVDN` (0x6D)

`LDAP_RES_COMPARE` (0x6F)

`LDAP_RES_EXTENDED` (0x78) -- new in LDAPv3

`ldap_result()` returns 0 if the timeout expired and -1 if an error occurs, in which case the error parameters of the LDAP session handle will be set accordingly.

`ldap_msgfree()` frees each message in the result chain pointed to by `res` and returns the type of the last message in the chain. If `res` is NULL, then nothing is done and the value zero is returned.

`ldap_msgtype()` returns the type of the LDAP message it is passed as a parameter. The type will be one of the types listed previously, or -1 on error.

`ldap_msgid()` returns the message ID associated with the LDAP message passed as a parameter, or `-1` on error.

Handling Errors and Parsing Results

Use the functions in this section to extract information from results and to handle errors returned by other LDAP API routines.

`ldap_parse_result`, `ldap_parse_sasl_bind_result`, `ldap_parse_extended_result`, and `ldap_err2string`

Note that `ldap_parse_sasl_bind_result()` and `ldap_parse_extended_result()` must typically be used in addition to `ldap_parse_result()` to retrieve all the result information from SASL Bind and Extended Operations respectively.

The `ldap_parse_result()`, `ldap_parse_sasl_bind_result()`, and `ldap_parse_extended_result()` functions all skip over messages of type `LDAP_RES_SEARCH_ENTRY` and `LDAP_RES_SEARCH_REFERENCE` when looking for a result message to parse. They return the constant `LDAP_SUCCESS` if the result was successfully parsed and another LDAP error code if not. Note that the LDAP error code that indicates the outcome of the operation performed by the server is placed in the `errcodep` `ldap_parse_result()` parameter. If a chain of messages that contains more than one result message is passed to these routines they always operate on the first result in the chain.

`ldap_err2string()` is used to convert a numeric LDAP error code, as returned by `ldap_parse_result()`, `ldap_parse_sasl_bind_result()`, `ldap_parse_extended_result()` or one of the synchronous API operation calls, into an informative zero-terminated character string message describing the error. It returns a pointer to static data.

Syntax

```
int ldap_parse_result
(
    LDAP          *ld,
    LDAPMessage   *res,
    int           *errcodep,
    char          **matcheddn,
    char          **errmsgp,
    char          ***referralsp,
    LDAPControl   ***serverctrlsp,
    int           freeit
);

int ldap_parse_sasl_bind_result
(
    LDAP          *ld,
    LDAPMessage   *res,
    struct berval **servercredp,
    int           freeit
);

int ldap_parse_extended_result
(
    LDAP          *ld,
    LDAPMessage   *res,
    char          **retoidp,
```

```

struct berval  **retdatap,
int            freeit
);
#define LDAP_NOTICE_OF_DISCONNECTION  "1.3.6.1.4.1.1466.20036"
char *ldap_err2string( int err );

```

The routines immediately following are deprecated. To learn more about them, see RFC 1823.

```

int ldap_result2error
(
LDAP          *ld,
LDAPMessage   *res,
int           freeit
);
void ldap_perror( LDAP *ld, const char *msg );

```

Parameters

Table 11–21 lists and describes parameters for handling errors and parsing results.

Table 11–21 Parameters for Handling Errors and Parsing Results

Parameter	Description
ld	The session handle.
res	The result of an LDAP operation as returned by <code>ldap_result()</code> or one of the synchronous API operation calls.
errcodep	This result parameter will be filled in with the LDAP error code field from the <code>LDAPMessage</code> message. This is the indication from the server of the outcome of the operation. <code>NULL</code> should be passed to ignore this field.
matcheddn	In the case of a return of <code>LDAP_NO_SUCH_OBJECT</code> , this result parameter will be filled in with a DN indicating how much of the name in the request was recognized. <code>NULL</code> should be passed to ignore this field. The matched DN string should be freed by calling <code>ldap_memfree()</code> which is described later in this document.
errmsgp	This result parameter will be filled in with the contents of the error message field from the <code>LDAPMessage</code> message. The error message string should be freed by calling <code>ldap_memfree()</code> which is described later in this document. <code>NULL</code> should be passed to ignore this field.
referralsp	This result parameter will be filled in with the contents of the referrals field from the <code>LDAPMessage</code> message, indicating zero or more alternate LDAP servers where the request is to be retried. The referrals array should be freed by calling <code>ldap_value_free()</code> which is described later in this document. <code>NULL</code> should be passed to ignore this field.
serverctrlsp	This result parameter will be filled in with an allocated array of controls copied out of the <code>LDAPMessage</code> message. The control array should be freed by calling <code>ldap_controls_free()</code> which was described earlier.
freeit	A Boolean that determines whether the <code>res</code> parameter is disposed of or not. Pass any nonzero value to have these routines free <code>res</code> after extracting the requested information. This is provided as a convenience; you can also use <code>ldap_msgfree()</code> to free the result later. If <code>freeit</code> is nonzero, the entire chain of messages represented by <code>res</code> is disposed of.
servercredp	For SASL bind results, this result parameter will be filled in with the credentials passed back by the server for mutual authentication, if given. An allocated <code>berval</code> structure is returned that should be disposed of by calling <code>ber_bvfree()</code> . <code>NULL</code> should be passed to ignore this field.

Table 11–21 (Cont.) Parameters for Handling Errors and Parsing Results

Parameter	Description
retoidp	For extended results, this result parameter will be filled in with the dotted-OID text representation of the name of the extended operation response. This string should be disposed of by calling <code>ldap_memfree()</code> . <code>NULL</code> should be passed to ignore this field. The <code>LDAP_NOTICE_OF_DISCONNECTION</code> macro is defined as a convenience for clients that wish to check an OID to see if it matches the one used for the unsolicited Notice of Disconnection (defined in RFC 2251[2] section 4.4.1).
retdatap	For extended results, this result parameter will be filled in with a pointer to a <code>struct berval</code> containing the data in the extended operation response. It should be disposed of by calling <code>ber_bvfree()</code> . <code>NULL</code> should be passed to ignore this field.
err	For <code>ldap_err2string()</code> , an LDAP error code, as returned by <code>ldap_parse_result()</code> or another LDAP API call.

Usage Notes

See RFC 1823 for a description of parameters peculiar to the deprecated routines.

Stepping Through a List of Results

Use the routines in this section to step through the list of messages in a result chain returned by `ldap_result()`.

ldap_first_message and ldap_next_message

The result chain for search operations can include referral messages, entry messages, and result messages.

`ldap_count_messages()` is used to count the number of messages returned. The `ldap_msgtype()` function, described previously, can be used to distinguish between the different message types.

```
LDAPMessage *ldap_first_message( LDAP *ld, LDAPMessage *res );
LDAPMessage *ldap_next_message( LDAP *ld, LDAPMessage *msg );
int ldap_count_messages( LDAP *ld, LDAPMessage *res );
```

Parameters

[Table 11–22](#) lists and describes the parameters for stepping through a list of results.

Table 11–22 Parameters for Stepping Through a List of Results

Parameter	Description
ld	The session handle.
res	The result chain, as obtained by a call to one of the synchronous search routines or <code>ldap_result()</code> .
msg	The message returned by a previous call to <code>ldap_first_message()</code> or <code>ldap_next_message()</code> .

Usage Notes

`ldap_first_message()` and `ldap_next_message()` will return `NULL` when no more messages exist in the result set to be returned. `NULL` is also returned if an error occurs while stepping through the entries, in which case the error parameters in the session handle `ld` will be set to indicate the error.

If successful, `ldap_count_messages()` returns the number of messages contained in a chain of results; if an error occurs such as the `res` parameter being invalid, `-1` is returned. The `ldap_count_messages()` call can also be used to count the number of messages that remain in a chain if called with a message, entry, or reference returned by `ldap_first_message()`, `ldap_next_message()`, `ldap_first_entry()`, `ldap_next_entry()`, `ldap_first_reference()`, `ldap_next_reference()`.

Parsing Search Results

Use the functions in this section to parse the entries and references returned by `ldap_search` functions. These results are returned in an opaque structure that may be accessed by calling the routines described in this section. Routines are provided to step through the entries and references returned, step through the attributes of an entry, retrieve the name of an entry, and retrieve the values associated with a given attribute in an entry.

`ldap_first_entry`, `ldap_next_entry`, `ldap_first_reference`, `ldap_next_reference`, `ldap_count_entries`, and `ldap_count_references`

The `ldap_first_entry()` and `ldap_next_entry()` routines are used to step through and retrieve the list of entries from a search result chain. The `ldap_first_reference()` and `ldap_next_reference()` routines are used to step through and retrieve the list of continuation references from a search result chain. `ldap_count_entries()` is used to count the number of entries returned. `ldap_count_references()` is used to count the number of references returned.

```
LDAPMessage *ldap_first_entry( LDAP *ld, LDAPMessage *res );
LDAPMessage *ldap_next_entry( LDAP *ld, LDAPMessage *entry );
LDAPMessage *ldap_first_reference( LDAP *ld, LDAPMessage *res );
LDAPMessage *ldap_next_reference( LDAP *ld, LDAPMessage *ref );
int ldap_count_entries( LDAP *ld, LDAPMessage *res );
int ldap_count_references( LDAP *ld, LDAPMessage *res );
```

Parameters

[Table 11-23](#) lists and describes the parameters for retrieving entries and continuation references from a search result chain, and for counting entries returned.

Table 11-23 Parameters for Retrieving Entries and Continuation References from a Search Result Chain, and for Counting Entries Returned

Parameter	Description
<code>ld</code>	The session handle.
<code>res</code>	The search result, as obtained by a call to one of the synchronous search routines or <code>ldap_result()</code> .
<code>entry</code>	The entry returned by a previous call to <code>ldap_first_entry()</code> or <code>ldap_next_entry()</code> .
<code>ref</code>	The reference returned by a previous call to <code>ldap_first_reference()</code> or <code>ldap_next_reference()</code> .

Usage Notes

`ldap_first_entry()`, `ldap_next_entry()`, `ldap_first_reference()`, and `ldap_next_reference()` all return `NULL` when no more entries or references exist in the result set to be returned. `NULL` is also returned if an error occurs while stepping

through the entries or references, in which case the error parameters in the session handle `ld` will be set to indicate the error.

`ldap_count_entries()` returns the number of entries contained in a chain of entries; if an error occurs such as the `res` parameter being invalid, `-1` is returned. The `ldap_count_entries()` call can also be used to count the number of entries that remain in a chain if called with a message, entry or reference returned by `ldap_first_message()`, `ldap_next_message()`, `ldap_first_entry()`, `ldap_next_entry()`, `ldap_first_reference()`, `ldap_next_reference()`.

`ldap_count_references()` returns the number of references contained in a chain of search results; if an error occurs such as the `res` parameter being invalid, `-1` is returned. The `ldap_count_references()` call can also be used to count the number of references that remain in a chain.

ldap_first_attribute and ldap_next_attribute

Use the functions in this section to step through the list of attribute types returned with an entry.

Syntax

```
char *ldap_first_attribute
(
    LDAP          *ld,
    LDAPMessage   *entry,
    BerElement    **ptr
);

char *ldap_next_attribute
(
    LDAP          *ld,
    LDAPMessage   *entry,
    BerElement    *ptr
);

void ldap_memfree( char *mem );
```

Parameters

[Table 11-24](#) lists and describes the parameters for stepping through attribute types returned with an entry.

Table 11-24 Parameters for Stepping Through Attribute Types Returned with an Entry

Parameter	Description
<code>ld</code>	The session handle.
<code>entry</code>	The entry whose attributes are to be stepped through, as returned by <code>ldap_first_entry()</code> or <code>ldap_next_entry()</code> .
<code>ptr</code>	In <code>ldap_first_attribute()</code> , the address of a pointer used internally to keep track of the current position in the entry. In <code>ldap_next_attribute()</code> , the pointer returned by a previous call to <code>ldap_first_attribute()</code> . The <code>BerElement</code> type itself is an opaque structure.
<code>mem</code>	A pointer to memory allocated by the LDAP library, such as the attribute type names returned by <code>ldap_first_attribute()</code> and <code>ldap_next_attribute()</code> , or the DN returned by <code>ldap_get_dn()</code> . If <code>mem</code> is <code>NULL</code> , the <code>ldap_memfree()</code> call does nothing.

Usage Notes

`ldap_first_attribute()` and `ldap_next_attribute()` returns `NULL` when the end of the attributes is reached, or if there is an error. In the latter case, the error parameters in the session handle `ld` are set to indicate the error.

Both routines return a pointer to an allocated buffer containing the current attribute name. This should be freed when no longer in use by calling `ldap_memfree()`.

`ldap_first_attribute()` will allocate and return in `ptr` a pointer to a `BerElement` used to keep track of the current position. This pointer may be passed in subsequent calls to `ldap_next_attribute()` to step through the entry's attributes. After a set of calls to `ldap_first_attribute()` and `ldap_next_attribute()`, if `ptr` is non-null, it should be freed by calling `ber_free(ptr, 0)`. Note that it is very important to pass the second parameter as 0 (zero) in this call, since the buffer associated with the `BerElement` does not point to separately allocated memory.

The attribute type names returned are suitable for passing in a call to `ldap_get_values()` and friends to retrieve the associated values.

`ldap_get_values`, `ldap_get_values_len`, `ldap_count_values`, `ldap_count_values_len`, `ldap_value_free`, and `ldap_value_free_len`

`ldap_get_values()` and `ldap_get_values_len()` are used to retrieve the values of a given attribute from an entry. `ldap_count_values()` and `ldap_count_values_len()` are used to count the returned values.

`ldap_value_free()` and `ldap_value_free_len()` are used to free the values.

Syntax

```
char **ldap_get_values
(
    LDAP          *ld,
    LDAPMessage   *entry,
    const char    *attr
);

struct berval **ldap_get_values_len
(
    LDAP          *ld,
    LDAPMessage   *entry,
    const char    *attr
);

int ldap_count_values( char **vals );
int ldap_count_values_len( struct berval **vals );
void ldap_value_free( char **vals );
void ldap_value_free_len( struct berval **vals );
```

Parameters

[Table 11-25](#) lists and describes the parameters for retrieving and counting attribute values.

Table 11-25 Parameters for Retrieving and Counting Attribute Values

Parameter	Description
<code>ld</code>	The session handle.
<code>entry</code>	The entry from which to retrieve values, as returned by <code>ldap_first_entry()</code> or <code>ldap_next_entry()</code> .

Table 11–25 (Cont.) Parameters for Retrieving and Counting Attribute Values

Parameter	Description
attr	The attribute whose values are to be retrieved, as returned by <code>ldap_first_attribute()</code> or <code>ldap_next_attribute()</code> , or a caller-supplied string (for example, "mail").
vals	The values returned by a previous call to <code>ldap_get_values()</code> or <code>ldap_get_values_len()</code> .

Usage Notes

Two forms of the various calls are provided. The first form is only suitable for use with non-binary character string data. The second `_len` form is used with any kind of data.

`ldap_get_values()` and `ldap_get_values_len()` return `NULL` if no values are found for `attr` or if an error occurs.

`ldap_count_values()` and `ldap_count_values_len()` return `-1` if an error occurs such as the `vals` parameter being invalid.

If a `NULL` `vals` parameter is passed to `ldap_value_free()` or `ldap_value_free_len()`, nothing is done.

Note that the values returned are dynamically allocated and should be freed by calling either `ldap_value_free()` or `ldap_value_free_len()` when no longer in use.

ldap_get_dn, ldap_explode_dn, ldap_explode_rdn, and ldap_dn2ufn

`ldap_get_dn()` is used to retrieve the name of an entry. `ldap_explode_dn()` and `ldap_explode_rdn()` are used to break up a name into its component parts. `ldap_dn2ufn()` is used to convert the name into a more user friendly format.

Syntax

```
char *ldap_get_dn( LDAP *ld, LDAPMessage *entry );
char **ldap_explode_dn( const char *dn, int notypes );
char **ldap_explode_rdn( const char *rdn, int notypes );
char *ldap_dn2ufn( const char *dn );
```

Parameters

[Table 11–26](#) lists and describes the parameters for retrieving, exploding, and converting entry names.

Table 11–26 Parameters for Retrieving, Exploding, and Converting Entry Names

Parameter	Description
ld	The session handle.
entry	The entry whose name is to be retrieved, as returned by <code>ldap_first_entry()</code> or <code>ldap_next_entry()</code> .
dn	The DN to explode, such as returned by <code>ldap_get_dn()</code> .
rdn	The RDN to explode, such as returned in the components of the array returned by <code>ldap_explode_dn()</code> .
notypes	A Boolean parameter, if nonzero indicating that the DN or RDN components are to have their type information stripped off: <code>cn=Babs</code> would become <code>Babs</code> .

Usage Notes

`ldap_get_dn()` returns `NULL` if a DN parsing error occurs. The function sets error parameters in the session handle `ld` to indicate the error. It returns a pointer to newly allocated space that the caller should free by calling `ldap_memfree()` when it is no longer in use.

`ldap_explode_dn()` returns a `NULL`-terminated `char *` array containing the RDN components of the DN supplied, with or without types as indicated by the `notypes` parameter. The components are returned in the order they appear in the DN. The array returned should be freed when it is no longer in use by calling `ldap_value_free()`.

`ldap_explode_rdn()` returns a `NULL`-terminated `char *` array containing the components of the RDN supplied, with or without types as indicated by the `notypes` parameter. The components are returned in the order they appear in the `rdn`. The array returned should be freed when it is no longer in use by calling `ldap_value_free()`.

`ldap_dn2ufn()` converts the DN into a user friendly format. The UFN returned is newly allocated space that should be freed by a call to `ldap_memfree()` when no longer in use.

`ldap_get_entry_controls`

`ldap_get_entry_controls()` is used to extract LDAP controls from an entry.

Syntax

```
int ldap_get_entry_controls
(
    LDAP          *ld,
    LDAPMessage   *entry,
    LDAPControl   ***serverctrlsp
);
```

Parameters

[Table 11-27](#) lists and describes the parameters for extracting LDAP control from an entry.

Table 11-27 *Parameters for Extracting LDAP Controls from an Entry*

Parameters	Description
<code>ld</code>	The session handle.
<code>entry</code>	The entry to extract controls from, as returned by <code>ldap_first_entry()</code> or <code>ldap_next_entry()</code> .
<code>serverctrlsp</code>	This result parameter will be filled in with an allocated array of controls copied out of entry. The control array should be freed by calling <code>ldap_controls_free()</code> . If <code>serverctrlsp</code> is <code>NULL</code> , no controls are returned.

Usage Notes

`ldap_get_entry_controls()` returns an LDAP error code that indicates whether the reference could be successfully parsed (`LDAP_SUCCESS` if all goes well).

`ldap_parse_reference`

Use `ldap_parse_reference()` to extract referrals and controls from a `SearchResultReference` message.

Syntax

```
int ldap_parse_reference
(
LDAP          *ld,
LDAPMessage  *ref,
char          ***referralsp,
LDAPControl  ***serverctrlsp,
int          freeit
);
```

Parameters

[Table 11–28](#) lists and describes parameters for extracting referrals and controls from a `SearchResultReference` message.

Table 11–28 Parameters for Extracting Referrals and Controls from a SearchResultReference Message

Parameter	Description
<code>ld</code>	The session handle.
<code>ref</code>	The reference to parse, as returned by <code>ldap_result()</code> , <code>ldap_first_reference()</code> , or <code>ldap_next_reference()</code> .
<code>referralsp</code>	This result parameter will be filled in with an allocated array of character strings. The elements of the array are the referrals (typically LDAP URLs) contained in <code>ref</code> . The array should be freed when no longer in used by calling <code>ldap_value_free()</code> . If <code>referralsp</code> is <code>NULL</code> , the referral URLs are not returned.
<code>serverctrlsp</code>	This result parameter will be filled in with an allocated array of controls copied out of <code>ref</code> . The control array should be freed by calling <code>ldap_controls_free()</code> . If <code>serverctrlsp</code> is <code>NULL</code> , no controls are returned.
<code>freeit</code>	A Boolean that determines whether the <code>ref</code> parameter is disposed of or not. Pass any nonzero value to have this routine free <code>ref</code> after extracting the requested information. This is provided as a convenience. You can also use <code>ldap_msgfree()</code> to free the result later.

Usage Notes

`ldap_parse_reference()` returns an LDAP error code that indicates whether the reference could be successfully parsed (`LDAP_SUCCESS` if all goes well).

Sample C API Usage

The following examples show how to use the C API both with and without SSL and for SASL authentication. More complete examples are given in RFC 1823. The sample code for the command-line tool to perform an LDAP search also demonstrates use of the API in both the SSL and the non-SSL mode.

This section contains these topics:

- [C API Usage with SSL](#)
- [C API Usage Without SSL](#)
- [C API Usage for SASL-Based DIGEST-MD5 Authentication](#)

C API Usage with SSL

```
#include <stdio.h>
#include <ldap.h>
```

```

main()
{
LDAP      *ld;
int       ret = 0;
...
/* open a connection */
if ((ld = ldap_open("MyHost", 636)) == NULL)
    exit( 1 );

/* SSL initialization */
ret = ldap_init_SSL(&ld->ld_sb, "file:/sslwallet", "welcome",
                  GSLC_SSL_ONEWAY_AUTH );

if(ret != 0)
{
printf(" %s \n", ldap_err2string(ret));
exit(1);
}

/* authenticate as nobody */
if ( ldap_bind_s( ld, NULL, NULL ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_bind_s" );
    exit( 1 );
}

.
.
.
}

```

Because the user is making the `ldap_init_SSL` call, the client/server communication in the previous example is secured by using SSL.

C API Usage Without SSL

```

#include <stdio.h>
#include <ldap.h>

main()
{
LDAP      *ld;
int       ret = 0;
.
.
.
/* open a connection */
if ( (ld = ldap_open( "MyHost", LDAP_PORT
)) == NULL )
    exit( 1 );

/* authenticate as nobody */
if ( ldap_bind_s( ld, NULL, NULL ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_bind_s" );
    exit( 1 );
}

.
.
.
}

```

In the previous example, the user is not making the `ldap_init_SSL` call, and the client-to-server communication is therefore not secure.

C API Usage for SASL-Based DIGEST-MD5 Authentication

This sample program illustrates the usage of LDAP SASL C-API for SASL-based DIGEST-MD5 authentication to a directory server.

```

/*
EXPORT FUNCTION(S)
    NONE

INTERNAL FUNCTION(S)
    NONE

STATIC FUNCTION(S)
    NONE

NOTES
Usage:
    saslbind -h ldap_host -p ldap_port -D authentication_identity_dn \
            -w password
options
    -h    LDAP host
    -p    LDAP port
    -D    DN of the identity for authentication
    -w    Password

Default SASL authentication parameters used by the demo program
SASL Security Property :    Currently only "auth" security property
                             is supported by the C-API. This demo
                             program uses this security property.

SASL Mechanism          :    Supported mechanisms by OID
                             "DIGEST-MD5" - This demo program
                             illustrates it's usage.
                             "EXTERNAL" - SSL authentication is used.
                             (This demo program does
                             not illustrate it's usage.)

Authorization identity :    This demo program does not use any
                             authorization identity.

MODIFIED    (MM/DD/YY)
*****    06/12/03 - Creation

*/

/*-----
PRIVATE TYPES AND CONSTANTS
-----*/

/*-----
STATIC FUNCTION DECLARATIONS
-----*/

#include <stdio.h>
#include <stdlib.h>
#include <ldap.h>

static int ldap_version = LDAP_VERSION3;

```



```

main (int argc, char **argv)
{
    LDAP*          ld;
    extern char*   optarg;
    char*          ldap_host = NULL;
    char*          ldap_bind_dn = NULL;
    char*          ldap_bind_pw = NULL;
    int            authmethod = 0;
    char           ldap_local_host[256] = "localhost";
    int            ldap_port = 389;
    char*          authcid = (char *)NULL;
    char*          mech = "DIGEST-MD5"; /* SASL mechanism */
    char*          authzid = (char *)NULL;
    char*          sasl_secprops = "auth";
    char*          realm = (char *)NULL;
    int            status = LDAP_SUCCESS;
    OraLdapHandle  sasl_cred = (OraLdapHandle )NULL;
    OraLdapClientCtx *cctx = (OraLdapClientCtx *)NULL;
    int            i = 0;

    while (( i = getopt( argc, argv,
        "D:h:p:w:E:P:U:V:W:O:R:X:Y:Z"
        )) != EOF ) {
    switch( i ) {

    case 'h':/* ldap host */
        ldap_host = (char *)strdup( optarg );
        break;
    case 'D':/* bind DN */
        authcid = (char *)strdup( optarg );
        break;

    case 'p':/* ldap port */
        ldap_port = atoi( optarg );
        break;
    case 'w':/* Password */
        ldap_bind_pw = (char *)strdup( optarg );
        break;

        default:
            printf("Invalid Arguments passed\n" );
    }
    }

    /* Get the connection to the LDAP server */
    if (ldap_host == NULL)
        ldap_host = ldap_local_host;

    if ((ld = ldap_open (ldap_host, ldap_port)) == NULL)
    {
        ldap_perror (ld, "ldap_init");
        exit (1);
    }

    /* Create the client context needed by LDAP C-API Oracle Extension functions*/
    status = ora_ldap_init_clientctx(&cctx);

```

```
if(LDAP_SUCCESS != status) {
    printf("Failed during creation of client context \n");
    exit(1);
}

/* Create SASL credentials */
sasl_cred = ora_ldap_create_cred_hdl(cctx, ORA_LDAP_CRED_HANDLE_SASL_MD5);

ora_ldap_set_cred_props(cctx, sasl_cred, ORA_LDAP_CRED_SASL_REALM,
    (void *)realm);
ora_ldap_set_cred_props(cctx, sasl_cred, ORA_LDAP_CRED_SASL_AUTH_PASSWORD,
    (void *)ldap_bind_pw);
ora_ldap_set_cred_props(cctx, sasl_cred, ORA_LDAP_CRED_SASL_AUTHORIZATION_ID,
    (void *)authzid);
ora_ldap_set_cred_props(cctx, sasl_cred, ORA_LDAP_CRED_SASL_SECURITY_PROPERTIES,
    (void *)sasl_secprops);

/* If connecting to the directory using SASL DIGEST-MD5, the Authentication ID
   has to be normalized before it's sent to the server,
   the LDAP C-API does this normalization based on the following flag set in
   SASL credential properties */
ora_ldap_set_cred_props(cctx, sasl_cred, ORA_LDAP_CRED_SASL_NORM_AUTHDN, (void
*)NULL);

/* SASL Authentication to LDAP Server */
status = (int)ora_ldap_init_SASL(cctx, ld, (char *)authcid, (char *)ORA_LDAP_
SASL_MECH_DIGEST_MD5,
    sasl_cred, NULL, NULL);

if(LDAP_SUCCESS == status) {
    printf("SASL bind successful \n" );
} else {
    printf("SASL bind failed with status : %d\n", status);
}

/* Free SASL Credentials */
ora_ldap_free_cred_hdl(cctx, sasl_cred);

status = ora_ldap_free_clientctx(cctx);

/* Unbind from LDAP server */
ldap_unbind (ld);

return (0);
}

/* end of file saslbnd.c */
```

Required Header Files and Libraries for the C API

To build applications with the C API, you need to:

- Include the header file located at `$ORACLE_HOME/ldap/public/ldap.h`.
- Dynamically link to the library located at
 - `$ORACLE_HOME/lib/libclntsh.so.10.1` on UNIX operating systems
 - `%ORACLE_HOME%\bin\oraldapclnt10.dll` on Windows operating systems

Dependencies and Limitations of the C API

This API can work against any release of Oracle Internet Directory. It requires either an Oracle environment or, at minimum, globalization support and other core libraries.

To use the different authentication modes in SSL, the directory server requires corresponding configuration settings.

See Also: *Oracle Internet Directory Administrator's Guide* for details about how to set the directory server in various SSL authentication modes

Oracle Wallet Manager is required for creating wallets if you are using the C API in SSL mode.

TCP/IP Socket Library is required.

The following Oracle libraries are required:

- Oracle SSL-related libraries
- Oracle system libraries

Sample libraries are included in the release for the sample command line tool. You should replace these libraries with your own versions of the libraries.

The product supports only those authentication mechanisms described in LDAP SDK specifications (RFC 1823).

DBMS_LDAP PL/SQL Reference

DBMS_LDAP contains the functions and procedures that enable PL/SQL programmers to access data from LDAP servers. This chapter examines all of the API functions in detail.

The chapter contains these topics:

- [Summary of Subprograms](#)
- [Exception Summary](#)
- [Data Type Summary](#)
- [Subprograms](#)

Note: Sample code for the DBMS_LDAP package is available at this URL:

http://www.oracle.com/technology/sample_code/

Look for the Oracle Identity Management link under Sample Applications–Oracle Application Server.

Summary of Subprograms

Table 12–1 DBMS_LDAP API Subprograms

Function or Procedure	Description
FUNCTION init	<code>init()</code> initializes a session with an LDAP server. This actually establishes a connection with the LDAP server.
FUNCTION simple_bind_s	The function <code>simple_bind_s()</code> can be used to perform simple user name and password authentication to the directory server.
FUNCTION bind_s	The function <code>bind_s()</code> can be used to perform complex authentication to the directory server.
FUNCTION unbind_s	The function <code>unbind_s()</code> is used for closing an active LDAP session.
FUNCTION compare_s	The function <code>compare_s()</code> can be used to test if a particular attribute in a particular entry has a particular value.
FUNCTION search_s	The function <code>search_s()</code> performs a synchronous search in the LDAP server. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is 'timed-out' by the server.

Table 12–1 (Cont.) DBMS_LDAP API Subprograms

Function or Procedure	Description
FUNCTION <code>search_st</code>	The function <code>search_st()</code> performs a synchronous search in the LDAP server with a client side time out. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is 'timed-out' by the client or the server.
FUNCTION <code>first_entry</code>	The function <code>first_entry</code> is used to retrieve the first entry in the result set returned by either <code>search_s()</code> or <code>search_st</code> .
FUNCTION <code>next_entry</code>	The function <code>next_entry()</code> is used to iterate to the next entry in the result set of a search operation.
FUNCTION <code>count_entries</code>	This function is used to count the number of entries in the result set. It can also be used to count the number of entries remaining during a traversal of the result set using a combination of the functions <code>first_entry()</code> and <code>next_entry</code> .
FUNCTION <code>first_attribute</code>	The function <code>first_attribute()</code> fetches the first attribute of a given entry in the result set.
FUNCTION <code>next_attribute</code>	The function <code>next_attribute()</code> fetches the next attribute of a given entry in the result set.
FUNCTION <code>get_dn</code>	The function <code>get_dn()</code> retrieves the X.500 distinguished name of a given entry in the result set.
FUNCTION <code>get_values</code>	The function <code>get_values()</code> can be used to retrieve all of the values associated with a given attribute in a given entry.
FUNCTION <code>get_values_len</code>	The function <code>get_values_len()</code> can be used to retrieve values of attributes that have a 'Binary' syntax.
FUNCTION <code>delete_s</code>	This function can be used to remove a leaf entry in the LDAP Directory Information Tree.
FUNCTION <code>modrdn2_s</code>	The function <code>modrdn2_s()</code> can be used to rename the relative distinguished name of an entry.
FUNCTION <code>err2string</code>	The function <code>err2string()</code> can be used to convert an LDAP error code to a string in the local language in which the API is operating.
FUNCTION <code>create_mod_array</code>	The function <code>create_mod_array()</code> allocates memory for array modification entries that will be applied to an entry using the <code>modify_s()</code> functions.
PROCEDURE <code>populate_mod_array (String Version)</code>	Populates one set of attribute information for add or modify operations. This procedure call has to happen after <code>DBMS_LDAP.create_mod_array()</code> is called.
PROCEDURE <code>populate_mod_array (Binary Version)</code>	Populates one set of attribute information for add or modify operations. This procedure call has to occur after <code>DBMS_LDAP.create_mod_array()</code> is called.
PROCEDURE <code>populate_mod_array (Binary Version. Uses BLOB Data Type)</code>	Populates one set of attribute information for add or modify operations. This procedure call has to happen after <code>DBMS_LDAP.create_mod_array()</code> is called.
FUNCTION <code>get_values_blob</code>	The function <code>get_values_blob()</code> can be used to retrieve larger values of attributes that have a binary syntax.
FUNCTION <code>count_values_blob</code>	Counts the number of values returned by <code>DBMS_LDAP.get_values_blob()</code> .
FUNCTION <code>value_free_blob</code>	Frees the memory associated with the <code>BLOB_COLLECTION</code> returned by <code>DBMS_LDAP.get_values_blob()</code> .

Table 12–1 (Cont.) DBMS_LDAP API Subprograms

Function or Procedure	Description
FUNCTION <code>modify_s</code>	Performs a synchronous modification of an existing LDAP directory entry. Before calling <code>add_s</code> , you must call <code>DBMS_LDAP.creat_mod_array()</code> and <code>DBMS_LDAP.populate_mod_array()</code> .
FUNCTION <code>add_s</code>	Adds a new entry to the LDAP directory synchronously. Before calling <code>add_s</code> , you must call <code>DBMS_LDAP.creat_mod_array()</code> and <code>DBMS_LDAP.populate_mod_array()</code> .
PROCEDURE <code>free_mod_array</code>	Frees the memory allocated by <code>DBMS_LDAP.create_mod_array()</code> .
FUNCTION <code>count_values</code>	Counts the number of values returned by <code>DBMS_LDAP.get_values()</code> .
FUNCTION <code>count_values_len</code>	Counts the number of values returned by <code>DBMS_LDAP.get_values_len()</code> .
FUNCTION <code>rename_s</code>	Renames an LDAP entry synchronously.
FUNCTION <code>explode_dn</code>	Breaks a DN up into its components.
FUNCTION <code>open_ssl</code>	Establishes an SSL (Secure Sockets Layer) connection over an existing LDAP connection.
FUNCTION <code>msgfree</code>	This function frees the chain of messages associated with the message handle returned by synchronous search functions.
FUNCTION <code>ber_free</code>	This function frees the memory associated with a handle to <code>BER_ELEMENT</code> .
FUNCTION <code>nls_convert_to_utf8</code>	The <code>nls_convert_to_utf8</code> function converts the input string containing database character set data to UTF8 character set data and returns it.
FUNCTION <code>nls_convert_from_utf8</code>	The <code>nls_convert_from_utf8</code> function converts the input string containing UTF8 character set data to database character set data and returns it.
FUNCTION <code>nls_get_dbcharset_name</code>	The <code>nls_get_dbcharset_name</code> function returns a string containing the database character set name.

See Also:

- "Searching the Directory" in Chapter 3 for more about `DBMS_LDAP.search_s()` and `DBMS_LDAP.search_st()`
- "Terminating the Session by Using DBMS_LDAP" in Chapter 3 for more about `DBMS_LDAP.unbind_s()`

Exception Summary

DBMS_LDAP can generate the exceptions described in [Table 12–2](#) on page 12-3.

Table 12–2 DBMS_LDAP Exception Summary

Exception Name	Oracle Error Number	Cause of Exception
<code>general_error</code>	31202	Raised anytime an error is encountered that does not have a specific PL/SQL exception associated with it. The error string contains the description of the problem in the user's language.

Table 12-2 (Cont.) DBMS_LDAP Exception Summary

Exception Name	Oracle Error Number	Cause of Exception
<code>init_failed</code>	31203	Raised by <code>DBMS_LDAP.init()</code> if there are problems.
<code>invalid_session</code>	31204	Raised by all functions and procedures in the <code>DBMS_LDAP</code> package if they are passed an invalid session handle.
<code>invalid_auth_method</code>	31205	Raised by <code>DBMS_LDAP.bind_s()</code> if the authentication method requested is not supported.
<code>invalid_search_scope</code>	31206	Raised by all search functions if the scope of the search is invalid.
<code>invalid_search_time_val</code>	31207	Raised by <code>DBMS_LDAP.search_st()</code> if it is given an invalid value for a time limit.
<code>invalid_message</code>	31208	Raised by all functions that iterate through a result-set for getting entries from a search operation if the message handle given to them is invalid.
<code>count_entry_error</code>	31209	Raised by <code>DBMS_LDAP.count_entries</code> if it cannot count the entries in a given result set.
<code>get_dn_error</code>	31210	Raised by <code>DBMS_LDAP.get_dn</code> if the DN of the entry it is retrieving is NULL.
<code>invalid_entry_dn</code>	31211	Raised by all functions that modify, add, or rename an entry if they are presented with an invalid entry DN.
<code>invalid_mod_array</code>	31212	Raised by all functions that take a modification array as an argument if they are given an invalid modification array.
<code>invalid_mod_option</code>	31213	Raised by <code>DBMS_LDAP.populate_mod_array</code> if the modification option given is anything other than <code>MOD_ADD</code> , <code>MOD_DELETE</code> or <code>MOD_REPLACE</code> .
<code>invalid_mod_type</code>	31214	Raised by <code>DBMS_LDAP.populate_mod_array</code> if the attribute type that is being modified is NULL.
<code>invalid_mod_value</code>	31215	Raised by <code>DBMS_LDAP.populate_mod_array</code> if the modification value parameter for a given attribute is NULL.
<code>invalid_rdn</code>	31216	Raised by all functions and procedures that expect a valid RDN and are provided with an invalid one.
<code>invalid_newparent</code>	31217	Raised by <code>DBMS_LDAP.rename_s</code> if the new parent of an entry being renamed is NULL.
<code>invalid_deleteoldrdn</code>	31218	Raised by <code>DBMS_LDAP.rename_s</code> if the <code>deleteoldrdn</code> parameter is invalid.
<code>invalid_notypes</code>	31219	Raised by <code>DBMS_LDAP.explode_dn</code> if the <code>notypes</code> parameter is invalid.
<code>invalid_ssl_wallet_loc</code>	31220	Raised by <code>DBMS_LDAP.open_ssl</code> if the wallet location is NULL but the SSL authentication mode requires a valid wallet.
<code>invalid_ssl_wallet_password</code>	31221	Raised by <code>DBMS_LDAP.open_ssl</code> if the wallet password given is NULL.
<code>invalid_ssl_auth_mode</code>	31222	Raised by <code>DBMS_LDAP.open_ssl</code> if the SSL authentication mode is not 1, 2 or 3.

Data Type Summary

The `DBMS_LDAP` package uses the data types described in [Table 12-3](#).

Table 12-3 *DBMS_LDAP Data Type Summary*

Data-Type	Purpose
<code>SESSION</code>	Used to hold the handle of the LDAP session. Nearly all of the functions in the API require a valid LDAP session to work.
<code>MESSAGE</code>	Used to hold a handle to the message retrieved from the result set. This is used by all functions that work with entry attributes and values.
<code>MOD_ARRAY</code>	Used to hold a handle to the array of modifications being passed to either <code>modify_s()</code> or <code>add_s()</code> .
<code>TIMEVAL</code>	Used to pass time limit information to the LDAP API functions that require a time limit.
<code>BER_ELEMENT</code>	Used to hold a handle to a BER structure used for decoding incoming messages.
<code>STRING_COLLECTION</code>	Used to hold a list of <code>VARCHAR2</code> strings that can be passed on to the LDAP server.
<code>BINVAL_COLLECTION</code>	Used to hold a list of <code>RAW</code> data, which represent binary data.
<code>BERVAL_COLLECTION</code>	Used to hold a list of <code>BERVAL</code> values that are used for populating a modification array.
<code>BLOB_COLLECTION</code>	Used to hold a list of <code>BLOB</code> data, which represent binary data.

Subprograms

This section takes a closer look at each of the `DBMS_LDAP` subprograms.

FUNCTION init

`init()` initializes a session with an LDAP server. This actually establishes a connection with the LDAP server.

Syntax

```
FUNCTION init
(
  hostname IN VARCHAR2,
  portnum  IN PLS_INTEGER
)
RETURN SESSION;
```

Parameters

Table 12-4 *INIT Function Parameters*

Parameter	Description
<code>hostname</code>	Contains a space-separated list of host names or dotted strings representing the IP address of hosts running an LDAP server to connect to. Each host name in the list may include a port number, which is separated from the host by a colon. The hosts are tried in the order listed, stopping with the first one to which a successful connection is made.

Table 12–4 (Cont.) INIT Function Parameters

Parameter	Description
portnum	Contains the TCP port number to connect to. If the port number is included with the host name, this parameter is ignored. If the parameter is not specified, and the host name does not contain the port number, a default port number of 389 is assumed.

Return Values**Table 12–5 INIT Function Return Values**

Value	Description
SESSION	A handle to an LDAP session that can be used for further calls to the API.

Exceptions**Table 12–6 INIT Function Exceptions**

Exception	Description
init_failed	Raised when there is a problem contacting the LDAP server.
general_error	For all other errors. The error string associated with the exception describes the error in detail.

Usage Notes

`DBMS_LDAP.init()` is the first function that should be called because it establishes a session with the LDAP server. Function `DBMS_LDAP.init()` returns a session handle, a pointer to an opaque structure that must be passed to subsequent calls pertaining to the session. This routine will return `NULL` and raise the `INIT_FAILED` exception if the session cannot be initialized. After `init()` has been called, the connection has to be authenticated using `DBMS_LDAP.bind_s` or `DBMS_LDAP.simple_bind_s()`.

See Also

`DBMS_LDAP.simple_bind_s()`, `DBMS_LDAP.bind_s()`.

FUNCTION simple_bind_s

The function `simple_bind_s` can be used to perform simple user name and password authentication to the directory server.

Syntax

```
FUNCTION simple_bind_s
(
  ld      IN SESSION,
  dn      IN VARCHAR2,
  passwd  IN VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 12–7 SIMPLE_BIND_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
dn	The Distinguished Name of the User that we are trying to login as.
passwd	A text string containing the password.

Return Values

Table 12–8 SIMPLE_BIND_S Function Return Values

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS on a successful completion. If there was a problem, one of the following exceptions will be raised.

Exceptions

Table 12–9 SIMPLE_BIND_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

DBMS_LDAP.simple_bind_s() can be used to authenticate a user whose directory distinguished name and directory password are known. It can be called only after a valid LDAP session handle is obtained from a call to DBMS_LDAP.init().

FUNCTION bind_s

The function bind_s can be used to perform complex authentication to the directory server.

Syntax

```
FUNCTION bind_s
(
  ld      IN SESSION,
  dn      IN VARCHAR2,
  cred    IN VARCHAR2,
  meth    IN PLS_INTEGER
)
RETURN PLS_INTEGER;
```

Parameters

Table 12–10 BIND_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
dn	The distinguished name of the user.

Table 12–10 (Cont.) BIND_S Function Parameters

Parameter	Description
cred	A text string containing the credentials used for authentication.
meth	The authentication method.

Return Values**Table 12–11 BIND_S Function Return Values**

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS upon successful completion. One of the following exceptions is raised if there is a problem.

Exceptions**Table 12–12 BIND_S Function Exceptions**

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_auth_method	Raised if the authentication method requested is not supported.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

DBMS_LDAP.bind_s() can be used to authenticate a user. It can be called only after a valid LDAP session handle is obtained from a call to DBMS_LDAP.init().

See Also

DBMS_LDAP.init(), DBMS_LDAP.simple_bind_s().

FUNCTION unbind_s

The function unbind_s is used for closing an active LDAP session.

Syntax

```
FUNCTION unbind_s
(
  ld IN OUT SESSION
)
RETURN PLS_INTEGER;
```

Parameters**Table 12–13 UNBIND_S Function Parameters**

Parameter	Description
ld	A valid LDAP session handle.

Return Values

Table 12–14 UNBIND_S Function Return Values

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS on proper completion. One of the following exceptions is raised otherwise.

Exceptions

Table 12–15 UNBIND_S Function Exceptions

Exception	Description
invalid_session	Raised if the sessions handle ld is invalid.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

The `unbind_s()` function sends an unbind request to the server, closes all open connections associated with the LDAP session, and disposes of all resources associated with the session handle before returning. After a call to this function, the session handle `ld` is invalid.

See Also

`DBMS_LDAP.bind_s()`, `DBMS_LDAP.simple_bind_s()`.

FUNCTION compare_s

The function `compare_s` can be used to test if a particular attribute in a particular entry has a particular value.

Syntax

```
FUNCTION compare_s
(
  ld      IN SESSION,
  dn      IN VARCHAR2,
  attr    IN VARCHAR2,
  value   IN VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 12–16 COMPARE_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
dn	The name of the entry to compare against.
attr	The attribute to compare against.
value	A string attribute value to compare against.

Return Values

Table 12–17 COMPARE_S Function Return Values

Value	Description
PLS_INTEGER	COMPARE_TRUE if the given attribute has a matching value. COMPARE_FALSE if the given attribute does not have a matching value.

Exceptions

Table 12–18 COMPARE_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

The function `compare_s` can be used to assert that an attribute in the directory has a certain value. This operation can be performed only on attributes whose syntax enables them to be compared. The `compare_s` function can be called only after a valid LDAP session handle has been obtained from the `init()` function and authenticated by the `bind_s()` or `simple_bind_s()` functions.

See Also

DBMS_LDAP.bind_s()

FUNCTION search_s

The function `search_s` performs a synchronous search in the directory. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is timed out by the server.

Syntax

```
FUNCTION search_s
(
  ld      IN  SESSION,
  base    IN  VARCHAR2,
  scope   IN  PLS_INTEGER,
  filter  IN  VARCHAR2,
  attrs   IN  STRING_COLLECTION,
  attronly IN PLS_INTEGER,
  res     OUT MESSAGE
)
RETURN PLS_INTEGER;
```

Parameters

Table 12–19 SEARCH_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
base	The DN of the entry at which to start the search.

Table 12–19 (Cont.) SEARCH_S Function Parameters

Parameter	Description
scope	One of SCOPE_BASE (0x00), SCOPE_ONELEVEL (0x01), or SCOPE_SUBTREE (0x02), indicating the scope of the search.
filter	A character string representing the search filter. The value NULL can be passed to indicate that the filter "(objectclass=*)", which matches all entries, is to be used.
attrs	A collection of strings indicating which attributes to return for each matching entry. Passing NULL for this parameter causes all available user attributes to be retrieved. The special constant string NO_ATTRS ("1.1") may be used as the only string in the array to indicate that no attribute types are to be returned by the server. The special constant string ALL_USER_ATTRS ("*") can be used in the attrs array along with the names of some operational attributes to indicate that all user attributes plus the listed operational attributes are to be returned.
attrsonly	A boolean value that must be zero if both attribute types and values are to be returned, and nonzero if only types are wanted.
res	This is a result parameter that contains the results of the search upon completion of the call. If no results are returned, *res is set to NULL.

Return Values

Table 12–20 SEARCH_S Function Return Value

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS if the search operation succeeded. An exception is raised in all other cases.
res	If the search succeeded and there are entries, this parameter is set to a non-null value which can be used to iterate through the result set.

Exceptions

Table 12–21 SEARCH_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_search_scope	Raised if the search scope is not one of SCOPE_BASE, SCOPE_ONELEVEL, or SCOPE_SUBTREE.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

The function `search_s()` issues a search operation and does not return control to the user environment until all of the results have been returned from the server. Entries returned from the search, if any, are contained in the `res` parameter. This parameter is opaque to the caller. Entries, attributes, and values can be extracted by calling the parsing routines described in this chapter.

See Also

`DBMS_LDAP.search_st()`, `DBMS_LDAP.first_entry()`, `DBMS_LDAP.next_entry()`.

FUNCTION search_st

The function `search_st()` performs a synchronous search in the LDAP server with a client-side time out. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is timed out by the client or the server.

Syntax

```
FUNCTION search_st
(
  ld      IN  SESSION,
  base    IN  VARCHAR2,
  scope   IN  PLS_INTEGER,
  filter  IN  VARCHAR2,
  attrs   IN  STRING_COLLECTION,
  attronly IN PLS_INTEGER,
  tv      IN  TIMEVAL,
  res     OUT MESSAGE
)
RETURN PLS_INTEGER;
```

Parameters

Table 12-22 SEARCH_ST Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
base	The DN of the entry at which to start the search.
scope	One of <code>SCOPE_BASE</code> (0x00), <code>SCOPE_ONELEVEL</code> (0x01), or <code>SCOPE_SUBTREE</code> (0x02), indicating the scope of the search.
filter	A character string representing the search filter. The value <code>NULL</code> can be passed to indicate that the filter " <code>(objectclass=*)</code> ", which matches all entries, is to be used.
attrs	A collection of strings indicating which attributes to return for each matching entry. Passing <code>NULL</code> for this parameter causes all available user attributes to be retrieved. The special constant string <code>NO_ATTRS</code> (" <code>1.1</code> ") may be used as the only string in the array to indicate that no attribute types are to be returned by the server. The special constant string <code>ALL_USER_ATTRS</code> (" <code>*</code> ") can be used in the <code>attrs</code> array along with the names of some operational attributes to indicate that all user attributes plus the listed operational attributes are to be returned.
attrsonly	A boolean value that must be zero if both attribute types and values are to be returned, and nonzero if only types are wanted.
tv	The time out value, expressed in seconds and microseconds, that should be used for this search.
res	This is a result parameter which will contain the results of the search upon completion of the call. If no results are returned, <code>*res</code> is set to <code>NULL</code> .

Return Values

Table 12–23 *SEARCH_ST Function Return Values*

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS if the search operation succeeded. An exception is raised in all other cases.
res	If the search succeeded and there are entries, this parameter is set to a non-null value which can be used to iterate through the result set.

Exceptions

Table 12–24 *SEARCH_ST Function Exceptions*

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_search_scope	Raised if the search scope is not one of SCOPE_BASE, SCOPE_ONELEVEL or SCOPE_SUBTREE.
invalid_search_time_value	Raised if the time value specified for the time out is invalid.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

This function is very similar to DBMS_LDAP.search_s() except that it requires a time out value to be given.

See Also

DBMS_LDAP.search_s(), DBML_LDAP.first_entry(), DBMS_LDAP.next_entry.

FUNCTION first_entry

The function first_entry() is used to retrieve the first entry in the result set returned by either search_s() or search_st().

Syntax

```
FUNCTION first_entry
(
  ld IN SESSION,
  msg IN MESSAGE
)
RETURN MESSAGE;
```

Parameters

Table 12–25 *FIRST_ENTRY Function Parameters*

Parameter	Description
ld	A valid LDAP session handle.
msg	The search result, as obtained by a call to one of the synchronous search routines.

Return Values

Table 12–26 *FIRST_ENTRY* Return Values

Value	Description
MESSAGE	A handle to the first entry in the list of entries returned from the LDAP server. It is set to NULL if there was an error and an exception is raised.

Exceptions

Table 12–27 *FIRST_ENTRY* Exceptions

Exception	Description
invalid_session	Raised if the session handle <code>ld</code> is invalid.
invalid_message	Raised if the incoming <code>msg</code> handle is invalid.

Usage Notes

The function `first_entry()` should always be the first function used to retrieve the results from a search operation.

See Also

`DBMS_LDAP.next_entry()`, `DBMS_LDAP.search_s()`, `DBMS_LDAP.search_st()`

FUNCTION next_entry

The function `next_entry()` is used to iterate to the next entry in the result set of a search operation.

Syntax

```
FUNCTION next_entry
(
  ld IN SESSION,
  msg IN MESSAGE
)
RETURN MESSAGE;
```

Parameters

Table 12–28 *NEXT_ENTRY* Function Parameters

Parameter	Description
<code>ld</code>	A valid LDAP session handle.
<code>msg</code>	The search result, as obtained by a call to one of the synchronous search routines.

Return Values

Table 12–29 *NEXT_ENTRY* Function Return Values

Value	Description
MESSAGE	A handle to the next entry in the list of entries returned from the LDAP server. It is set to null if there was an error and an exception is raised.

Exceptions

Table 12–30 *NEXT_ENTRY* Function Exceptions

Exception	Description
invalid_session	Raised if the session handle, ld is invalid.
invalid_message	Raised if the incoming msg handle is invalid.

Usage Notes

The function `next_entry()` should always be called after a call to the function `first_entry()`. Also, the return value of a successful call to `next_entry()` should be used as `msg` argument used in a subsequent call to the function `next_entry()` to fetch the next entry in the list.

See Also

`DBMS_LDAP.first_entry()`, `DBMS_LDAP.search_s()`, `DBMS_LDAP.search_st()`

FUNCTION count_entries

This function is used to count the number of entries in the result set. It can also be used to count the number of entries remaining during a traversal of the result set using a combination of the functions `first_entry()` and `next_entry()`.

Syntax

```
FUNCTION count_entries
(
  ld IN SESSION,
  msg IN MESSAGE
)
RETURN PLS_INTEGER;
```

Parameters

Table 12–31 *COUNT_ENTRY* Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
msg	The search result, as obtained by a call to one of the synchronous search routines.

Return Values

Table 12–32 *COUNT_ENTRY Function Return Values*

Value	Description
PLS_INTEGER	Nonzero if there are entries in the result set. -1 if there was a problem.

Exceptions

Table 12–33 *COUNT_ENTRY Function Exceptions*

Exception	Description
invalid_session	Raised if the session handle <code>ld</code> is invalid.
invalid_message	Raised if the incoming <code>msg</code> handle is invalid.
count_entry_error	Raised if there was a problem in counting the entries.

Usage Notes

`count_entries()` returns the number of entries contained in a chain of entries; if an error occurs such as the `res` parameter being invalid, -1 is returned. The `count_entries()` call can also be used to count the number of entries that remain in a chain if called with a message, entry, or reference returned by `first_message()`, `next_message()`, `first_entry()`, `next_entry()`, `first_reference()`, `next_reference()`.

See Also

`DBMS_LDAP.first_entry()`, `DBMS_LDAP.next_entry()`.

FUNCTION first_attribute

The function `first_attribute()` fetches the first attribute of a given entry in the result set.

Syntax

```
FUNCTION first_attribute
(
  ld          IN  SESSION,
  ldapentry  IN  MESSAGE,
  ber_elem   OUT BER_ELEMENT
)
RETURN VARCHAR2;
```

Parameters

Table 12–34 *FIRST_ATTRIBUTE Function Parameters*

Parameter	Description
<code>ld</code>	A valid LDAP session handle.
<code>ldapentry</code>	The entry whose attributes are to be stepped through, as returned by <code>first_entry()</code> or <code>next_entry()</code> .
<code>ber_elem</code>	A handle to a <code>BER_ELEMENT</code> that is used to keep track of attributes in the entry that have already been read.

Return Values

Table 12–35 *FIRST_ATTRIBUTE Function Return Values*

Value	Description
VARCHAR2	The name of the attribute if it exists. NULL if no attribute exists or if an error occurred.
ber_elem	A handle used by <code>DBMS_LDAP.next_attribute()</code> to iterate over all of the attributes

Exceptions

Table 12–36 *FIRST_ATTRIBUTE Function Exceptions*

Exception	Description
<code>invalid_session</code>	Raised if the session handle <code>ld</code> is invalid.
<code>invalid_message</code>	Raised if the incoming <code>msg</code> handle is invalid.

Usage Notes

The handle to the `BER_ELEMENT` returned as a function parameter to `first_attribute()` should be used in the next call to `next_attribute()` to iterate through the various attributes of an entry. The name of the attribute returned from a call to `first_attribute()` can in turn be used in calls to the functions `get_values()` or `get_values_len()` to get the values of that particular attribute.

See Also

`DBMS_LDAP.next_attribute()`, `DBMS_LDAP.get_values()`, `DBMS_LDAP.get_values_len()`, `DBMS_LDAP.first_entry()`, `DBMS_LDAP.next_entry()`.

FUNCTION next_attribute

The function `next_attribute()` retrieves the next attribute of a given entry in the result set.

Syntax

```
FUNCTION next_attribute
(
  ld          IN SESSION,
  ldapentry   IN MESSAGE,
  ber_elem    IN BER_ELEMENT
)
RETURN VARCHAR2;
```

Parameters

Table 12–37 *NEXT_ATTRIBUTE Function Parameters*

Parameter	Description
<code>ld</code>	A valid LDAP session handle.
<code>ldapentry</code>	The entry whose attributes are to be stepped through, as returned by <code>first_entry()</code> or <code>next_entry()</code> .
<code>ber_elem</code>	A handle to a <code>BER_ELEMENT</code> that is used to keep track of attributes in the entry that have been read.

Return Values

Table 12–38 *NEXT_ATTRIBUTE Function Return Values*

Value	Description
VARCHAR2 (function return)	The name of the attribute if it exists.

Exceptions

Table 12–39 *NEXT_ATTRIBUTE Function Exceptions*

Exception	Description
invalid_session	Raised if the session handle <code>ld</code> is invalid.
invalid_message	Raised if the incoming <code>msg</code> handle is invalid.

Usage Notes

The handle to the `BER_ELEMENT` returned as a function parameter to `first_attribute()` should be used in the next call to `next_attribute()` to iterate through the various attributes of an entry. The name of the attribute returned from a call to `next_attribute()` can in turn be used in calls to the functions `get_values()` or `get_values_len()` to get the values of that particular attribute.

See Also

`DBMS_LDAP.first_attribute()`, `DBMS_LDAP.get_values()`, `DBMS_LDAP.get_values_len()`, `DBMS_LDAP.first_entry()`, `DBMS_LDAP.next_entry()`.

FUNCTION get_dn

The function `get_dn()` retrieves the X.500 distinguished name of given entry in the result set.

Syntax

```
FUNCTION get_dn
(
  ld IN SESSION,
  ldapentrymsg IN MESSAGE
)
RETURN VARCHAR2;
```

Parameters

Table 12–40 *GET_DN Function Parameters*

Parameter	Description
<code>ld</code>	A valid LDAP session handle.
<code>ldapentry</code>	The entry whose DN is to be returned.

Return Values

Table 12–41 *GET_DN Function Return Values*

Value	Description
VARCHAR2	The X.500 Distinguished name of the entry as a PL/SQL string. NULL if there was a problem.

Exceptions

Table 12–42 *GET_DN Function Exceptions*

Exception	Description
invalid_session	Raised if the session handle <code>ld</code> is invalid.
invalid_message	Raised if the incoming <code>msg</code> handle is invalid.
get_dn_error	Raised if there was a problem in determining the DN.

Usage Notes

The function `get_dn()` can be used to retrieve the DN of an entry as the program logic is iterating through the result set. This can in turn be used as an input to `explode_dn()` to retrieve the individual components of the DN.

See Also

`DBMS_LDAP.explode_dn()`.

FUNCTION `get_values`

The function `get_values()` can be used to retrieve all of the values associated with a given attribute in a given entry.

Syntax

```
FUNCTION get_values
(
  ld      IN SESSION,
  ldapentry IN MESSAGE,
  attr   IN VARCHAR2
)
RETURN STRING_COLLECTION;
```

Parameters

Table 12–43 *GET_VALUES Function Parameters*

Parameter	Description
<code>ld</code>	A valid LDAP session handle.
<code>ldapentry</code>	A valid handle to an entry returned from a search result.
<code>attr</code>	The name of the attribute for which values are being sought.

Return Values

Table 12–44 *GET_VALUES Function Return Values*

Value	Description
STRING_COLLECTION	A PL/SQL string collection containing all of the values of the given attribute. NULL if there are no values associated with the given attribute.

Exceptions

Table 12–45 *GET_VALUES Function Exceptions*

Exception	Description
invalid_session	Raised if the session handle <code>ld</code> is invalid.
invalid_message	Raised if the incoming entry handle is invalid.

Usage Notes

The function `get_values()` can only be called after the handle to entry has been first retrieved by call to either `first_entry()` or `next_entry()`. The name of the attribute may be known beforehand or can be determined by a call to `first_attribute()` or `next_attribute()`. The function `get_values()` always assumes that the data type of the attribute it is retrieving is a string. For retrieving binary data types, `get_values_len()` should be used.

See Also

`DBMS_LDAP.first_entry()`, `DBMS_LDAP.next_entry()`, `DBMS_LDAP.count_values()`, `DBMS_LDAP.get_values_len()`.

FUNCTION `get_values_len`

The function `get_values_len()` can be used to retrieve values of attributes that have a binary syntax.

Syntax

```
FUNCTION get_values_len
(
  ld      IN SESSION,
  ldapentry IN MESSAGE,
  attr    IN VARCHAR2
)
RETURN BINVAL_COLLECTION;
```

Parameters

Table 12–46 *GET_VALUES_LEN Function Parameters*

Parameter	Description
<code>ld</code>	A valid LDAP session handle.
<code>ldapentrymsg</code>	A valid handle to an entry returned from a search result.
<code>attr</code>	The string name of the attribute for which values are being sought.

Return Values

Table 12–47 *GET_VALUES_LEN Function Return Values*

Value	Description
BINVAL_COLLECTION	A PL/SQL 'Raw' collection containing all the values of the given attribute. NULL if there are no values associated with the given attribute.

Exceptions

Table 12–48 *GET_VALUES_LEN Function Exceptions*

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming entry handle is invalid.

Usage Notes

The function `get_values_len()` can only be called after the handle to an entry has been retrieved by a call to either `first_entry()` or `next_entry()`. The name of the attribute may be known beforehand or can also be determined by a call to `first_attribute()` or `next_attribute()`. This function can be used to retrieve both binary and non-binary attribute values.

See Also

`DBMS_LDAP.first_entry()`, `DBMS_LDAP.next_entry()`, `DBMS_LDAP.count_values_len()`, `DBMS_LDAP.get_values()`.

FUNCTION delete_s

The function `delete_s()` can be used to remove a leaf entry in the DIT.

Syntax

```
FUNCTION delete_s
(
  ld      IN SESSION,
  entrydn IN VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 12–49 *DELETE_S Function Parameters*

Parameter Name	Description
ld	A valid LDAP session.
entrydn	The X.500 distinguished name of the entry to delete.

Return Values**Table 12–50** *DELETE_S Function Return Values*

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS if the delete operation was successful. An exception is raised otherwise.

Exceptions**Table 12–51** *DELETE_S Function Exceptions*

Exception	Description
invalid_session	Raised if the session handle <code>ld</code> is invalid.
invalid_entry_dn	Raised if the distinguished name of the entry is invalid.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

The function `delete_s()` can be used to remove only leaf entries in the DIT. A leaf entry is an entry that does not have any entries under it. This function cannot be used to delete non-leaf entries.

See Also

DBMS_LDAP.modrdn2_s().

FUNCTION modrdn2_s

The function `modrdn2_s()` can be used to rename the relative distinguished name of an entry.

Syntax

```
FUNCTION modrdn2_s
(
  ld IN SESSION,
  entrydn IN VARCHAR2
  newrdn IN VARCHAR2
  deleteoldrdn IN PLS_INTEGER
)
RETURN PLS_INTEGER;
```

Parameters**Table 12–52** *MODRDN2_S Function Parameters*

Parameter	Description
<code>ld</code>	A valid LDAP session handle.
<code>entrydn</code>	The distinguished name of the entry (This entry must be a leaf node in the DIT).
<code>newrdn</code>	The new relative distinguished name of the entry.
<code>deleteoldrdn</code>	A boolean value that, if nonzero, indicates that the attribute values from the old name should be removed from the entry.

Return Values

Table 12–53 MODRDN2_S Function Return Values

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS if the operation was successful. An exception is raised otherwise.

Exceptions

Table 12–54 MODRDN2_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle <code>ld</code> is invalid.
invalid_entry_dn	Raised if the distinguished name of the entry is invalid.
invalid_rdn	Invalid LDAP RDN.
invalid_deleteoldrdn	Invalid LDAP deleteoldrdn.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

The function `nodrdn2_s()` can be used to rename the leaf nodes of a DIT. It simply changes the relative distinguished name by which they are known. The use of this function is being deprecated in the LDAP v3 standard. Please use `rename_s()`, which fulfills the same purpose.

See Also

DBMS_LDAP.rename_s().

FUNCTION err2string

The function `err2string()` can be used to convert an LDAP error code to a string in the local language in which the API is operating.

Syntax

```
FUNCTION err2string
(
  ldap_err IN PLS_INTEGER
)
RETURN VARCHAR2;
```

Parameters

Table 12–55 ERR2STRING Function Parameters

Parameter	Description
ldap_err	An error number returned from one of the API calls.

Return Values

Table 12–56 *ERR2STRING Function Return Values*

Value	Description
VARCHAR2	A character string translated to the local language. The string describes the error in detail.

Exceptions

`err2string()` raises no exceptions.

Usage Notes

In this release, the exception handling mechanism automatically invokes this function if any of the API calls encounter an error.

FUNCTION `create_mod_array`

The function `create_mod_array()` allocates memory for array modification entries that are applied to an entry using the `modify_s()` or `add_s()` functions.

Syntax

```

FUNCTION create_mod_array
(
  num IN PLS_INTEGER
)
RETURN MOD_ARRAY;
```

Parameters

Table 12–57 *CREATE_MOD_ARRAY Function Parameters*

Parameter	Description
<code>num</code>	The number of the attributes that you want to add or modify.

Return Values

Table 12–58 *CREATE_MOD_ARRAY Function Return Values*

Value	Description
<code>MOD_ARRAY</code>	The data structure holds a pointer to an LDAP mod array. Returns <code>NULL</code> if there was a problem.

Exceptions

`create_mod_array()` raises no exceptions.

Usage Notes

This function is one of the preparation steps for `DBMS_LDAP.add_s` and `DBMS_LDAP.modify_s`. It calls `DBMS_LDAP.free_mod_array` to free memory after the calls to `add_s` or `modify_s` have completed.

See Also

`DBMS_LDAP.populate_mod_array()`, `DBMS_LDAP.modify_s()`, `DBMS_LDAP.add_s()`, and `DBMS_LDAP.free_mod_array()`.

PROCEDURE populate_mod_array (String Version)

Populates one set of attribute information for add or modify operations.

Syntax

```
PROCEDURE populate_mod_array
(
  modptr    IN DBMS_LDAP.MOD_ARRAY,
  mod_op    IN PLS_INTEGER,
  mod_type  IN VARCHAR2,
  modval    IN DBMS_LDAP.STRING_COLLECTION
);
```

Parameters

Table 12–59 POPULATE_MOD_ARRAY (String Version) Procedure Parameters

Parameter	Description
modptr	The data structure holds a pointer to an LDAP mod array.
mod_op	This field specifies the type of modification to perform.
mod_type	This field indicates the name of the attribute type to which the modification applies.
modval	This field specifies the attribute values to add, delete, or replace. It is for string values only.

Exceptions

Table 12–60 POPULATE_MOD_ARRAY (String Version) Procedure Exceptions

Exception	Description
invalid_mod_array	Invalid LDAP mod array
invalid_mod_option	Invalid LDAP mod option
invalid_mod_type	Invalid LDAP mod type
invalid_mod_value	Invalid LDAP mod value

Usage Notes

This function is one of the preparation steps for `DBMS_LDAP.add_s` and `DBMS_LDAP.modify_s`. It has to happen after `DBMS_LDAP.create_mod_array` is called.

See Also

`DBMS_LDAP.create_mod_array()`, `DBMS_LDAP.modify_s()`, `DBMS_LDAP.add_s()`, and `DBMS_LDAP.free_mod_array()`.

PROCEDURE populate_mod_array (Binary Version)

Populates one set of attribute information for add or modify operations. This procedure call occurs after `DBMS_LDAP.create_mod_array()` is called.

Syntax

```
PROCEDURE populate_mod_array
(
  modptr    IN DBMS_LDAP.MOD_ARRAY,
  mod_op    IN PLS_INTEGER,
```

```

mod_type IN VARCHAR2,
modbval  IN DBMS_LDAP.BERVAL_COLLECTION
);

```

Parameters

Table 12–61 *POPULATE_MOD_ARRAY (Binary Version) Procedure Parameters*

Parameter	Description
modptr	This data structure holds a pointer to an LDAP mod array.
mod_op	This field specifies the type of modification to perform.
mod_type	This field indicates the name of the attribute type to which the modification applies.
modbval	This field specifies the attribute values to add, delete, or replace. It is for the binary values.

Exceptions

Table 12–62 *POPULATE_MOD_ARRAY (Binary Version) Procedure Exceptions*

Exception	Description
invalid_mod_array	Invalid LDAP mod array.
invalid_mod_option	Invalid LDAP mod option.
invalid_mod_type	Invalid LDAP mod type.
invalid_mod_value	Invalid LDAP mod value.

Usage Notes

This function is one of the preparation steps for `DBMS_LDAP.add_s` and `DBMS_LDAP.modify_s`. It is invoked after `DBMS_LDAP.create_mod_array` is called.

See Also

`DBMS_LDAP.create_mod_array()`, `DBMS_LDAP.modify_s()`, `DBMS_LDAP.add_s()`, and `DBMS_LDAP.free_mod_array()`.

PROCEDURE `populate_mod_array` (Binary Version. Uses BLOB Data Type)

Populates one set of attribute information for add or modify operations. This procedure call occurs after `DBMS_LDAP.create_mod_array()` is called.

Syntax

```

PROCEDURE populate_mod_array
(
  modptr IN DBMS_LDAP.MOD_ARRAY,
  mod_op IN PLS_INTEGER,
  mod_type IN VARCHAR2,
  modbval IN DBMS_LDAP.BLOB_COLLECTION
);

```

Parameters

Table 12–63 *POPULATE_MOD_ARRAY (Binary) Parameters*

Parameter	Description
modptr	This data structure holds a pointer to an LDAP mod array.
mod_op	This field specifies the type of modification to perform.
mod_type	This field indicates the name of the attribute type to which the modification applies.
modbval	This field specifies the binary attribute values to add, delete, or replace.

Exceptions

Table 12–64 *POPULATE_MOD_ARRAY (Binary) Exceptions*

Exception	Description
invalid_mod_array	Invalid LDAP mod array.
invalid_mod_option	Invalid LDAP mod option.
invalid_mod_type	Invalid LDAP mod type.
invalid_mod_value	Invalid LDAP mod value.

Usage Notes

This function is one of the preparation steps for `DBMS_LDAP.add_s` and `DBMS_LDAP.modify_s`. It is invoked after `DBMS_LDAP.create_mod_array` is called.

See Also

`DBMS_LDAP.create_mod_array()`, `DBMS_LDAP.modify_s()`, `DBMS_LDAP.add_s()`, and `DBMS_LDAP.free_mod_array()`.

FUNCTION `get_values_blob`

The function `get_values_blob()` can be used to retrieve larger values of attributes that have a binary syntax.

Syntax

```
Syntax
FUNCTION get_values_blob
(
  ld IN SESSION,
  ldapentry IN MESSAGE,
  attr IN VARCHAR2
)
RETURN BLOB_COLLECTION;
```

Parameters

Table 12–65 *GET_VALUES_BLOB Parameters*

Parameter	Description
ld	A valid LDAP session handle.
ldapentrymsg	A valid handle to an entry returned from a search result.

Table 12–65 (Cont.) GET_VALUES_BLOB Parameters

Parameter	Description
attr	The string name of the attribute for which values are being sought.

Return Values**Table 12–66 get_values_blob Return Values**

Value	Description
BLOB_COLLECTION	A PL/SQL BLOB collection containing all the values of the given attribute.
NULL	No values are associated with the given attribute.

Exceptions**Table 12–67 get_values_blob Exceptions**

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid message	Raised if the incoming entry handle is invalid.

Usage Notes

The function `get_values_blob()` can only be called after the handle to an entry has been retrieved by a call to either `first_entry()` or `next_entry()`. The name of the attribute may be known beforehand or can also be determined by a call to `first_attribute()` or `next_attribute()`. This function can be used to retrieve both binary and nonbinary attribute values.

See Also

`DBMS_LDAP.first_entry()`, `DBMS_LDAP.next_entry()`, `DBMS_LDAP.count_values_blob()`, `DBMS_LDAP.get_values()`.

FUNCTION count_values_blob

Counts the number of values returned by `DBMS_LDAP.get_values_blob()`.

Syntax

```
FUNCTION count_values_blob
(
  values IN DBMS_LDAP.BLOB_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters**Table 12–68 COUNT_VALUES_BLOB Parameters**

Parameter	Description
values	The collection of large binary values.

Return Values

Table 12–69 *COUNT_VALUES_BLOB Return Values*

Values	Description
PLS_INTEGER	Indicates the success or failure of the operation.

Exceptions

The function `count_values_blob()` raises no exceptions.

See Also

`DBMS_LDAP.count_values()`, `DBMS_LDAP.get_values_blob()`.

FUNCTION `value_free_blob`

Frees the memory associated with `BLOB_COLLECTION` returned by `DBMS_LDAP.get_values_blob()`.

Syntax

```
PROCEDURE value_free_blob
(
  vals IN OUT DBMS_LDAP.BLOB_COLLECTION
);
```

Parameters

Table 12–70 *VALUE_FREE_BLOB Parameters*

Parameter	Description
<code>vals</code>	The collection of large binary values returned by <code>DBMS_LDAP.get_values_blob()</code> .

Exceptions

`value_free_blob()` raises no exceptions.

See Also

`DBMS_LDAP.get_values_blob()`.

FUNCTION `modify_s`

Performs a synchronous modification of an existing LDAP directory entry.

Syntax

```
FUNCTION modify_s
(
  ld      IN DBMS_LDAP.SESSION,
  entrydn IN VARCHAR2,
  modptr  IN DBMS_LDAP.MOD_ARRAY
)
RETURN PLS_INTEGER;
```

Parameters

Table 12–71 *MODIFY_S Function Parameters*

Parameter	Description
ld	This parameter is a handle to an LDAP session returned by a successful call to <code>DBMS_LDAP.init()</code> .
entrydn	This parameter specifies the name of the directory entry whose contents are to be modified.
modptr	This parameter is the handle to an LDAP mod structure, as returned by successful call to <code>DBMS_LDAP.create_mod_array()</code> .

Return Values

Table 12–72 *MODIFY_S Function Return Values*

Value	Description
PLS_INTEGER	Indicates the success or failure of the modification operation.

Exceptions

Table 12–73 *MODIFY_S Function Exceptions*

Exception	Description
invalid_session	Invalid LDAP session.
invalid_entry_dn	Invalid LDAP entry dn.
invalid_mod_array	Invalid LDAP mod array.

Usage Notes

This function call has to follow successful calls of `DBMS_LDAP.create_mod_array()` and `DBMS_LDAP.populate_mod_array()`.

See Also

`DBMS_LDAP.create_mod_array()`, `DBMS_LDAP.populate_mod_array()`, `DBMS_LDAP.add_s()`, and `DBMS_LDAP.free_mod_array()`.

FUNCTION add_s

Adds a new entry to the LDAP directory synchronously. Before calling `add_s`, `DBMS_LDAP.create_mod_array()` and `DBMS_LDAP.populate_mod_array()` must be called.

Syntax

```
FUNCTION add_s
(
  ld      IN DBMS_LDAP.SESSION,
  entrydn IN VARCHAR2,
  modptr  IN DBMS_LDAP.MOD_ARRAY
)
RETURN PLS_INTEGER;
```

Parameters

Table 12–74 *ADD_S Function Parameters*

Parameter	Description
ld	This parameter is a handle to an LDAP session, as returned by a successful call to <code>DBMS_LDAP.init()</code> .
entrydn	This parameter specifies the name of the directory entry to be created.
modptr	This parameter is the handle to an LDAP mod structure, as returned by successful call to <code>DBMS_LDAP.create_mod_array()</code> .

Return Values

Table 12–75 *ADD_S Function Return Values*

Value	Description
PLS_INTEGER	Indicates the success or failure of the modification operation.

Exceptions

Table 12–76 *ADD_S Function Exceptions*

Exception	Description
invalid_session	Invalid LDAP session.
invalid_entry_dn	Invalid LDAP entry dn.
invalid_mod_array	Invalid LDAP mod array.

Usage Notes

The parent entry of the entry to be added must already exist in the directory. This function call has to follow successful calls to `DBMS_LDAP.create_mod_array()` and `DBMS_LDAP.populate_mod_array()`.

See Also

`DBMS_LDAP.create_mod_array()`, `DBMS_LDAP.populate_mod_array()`, `DBMS_LDAP.modify_s()`, and `DBMS_LDAP.free_mod_array()`.

PROCEDURE free_mod_array

Frees the memory allocated by `DBMS_LDAP.create_mod_array()`.

Syntax

```
PROCEDURE free_mod_array
(
  modptr IN DBMS_LDAP.MOD_ARRAY
);
```

Parameters**Table 12–77** *FREE_MOD_ARRAY Procedure Parameters*

Parameter	Description
modptr	This parameter is the handle to an LDAP mod structure returned by a successful call to <code>DBMS_LDAP.create_mod_array()</code> .

Exceptions

`free_mod_array` raises no exceptions.

See Also

`DBMS_LDAP.populate_mod_array()`, `DBMS_LDAP.modify_s()`, `DBMS_LDAP.add_s()`, and `DBMS_LDAP.create_mod_array()`.

FUNCTION count_values

Counts the number of values returned by `DBMS_LDAP.get_values()`.

Syntax

```
FUNCTION count_values
(
  values IN DBMS_LDAP.STRING_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters**Table 12–78** *COUNT_VALUES Function Parameters*

Parameter	Description
values	The collection of string values.

Return Values**Table 12–79** *COUNT_VALUES Function Return Values*

Value	Description
PLS_INTEGER	Indicates the success or failure of the operation.

Exceptions

`count_values` raises no exceptions.

See Also

`DBMS_LDAP.count_values_len()`, `DBMS_LDAP.get_values()`.

FUNCTION count_values_len

Counts the number of values returned by `DBMS_LDAP.get_values_len()`.

Syntax

```
FUNCTION count_values_len
(
```

```

values IN DBMS_LDAP.BINVAL_COLLECTION
)
RETURN PLS_INTEGER;

```

Parameters

Table 12–80 *COUNT_VALUES_LEN Function Parameters*

Parameter	Description
values	The collection of binary values.

Return Values

Table 12–81 *COUNT_VALUES_LEN Function Return Values*

Value	Description
PLS_INTEGER	Indicates the success or failure of the operation.

Exceptions

count_values_len raises no exceptions.

See Also

DBMS_LDAP.count_values(), DBMS_LDAP.get_values_len().

FUNCTION rename_s

Renames an LDAP entry synchronously.

Syntax

```

FUNCTION rename_s
(
ld          IN SESSION,
dn          IN VARCHAR2,
newrdn     IN VARCHAR2,
newparent  IN VARCHAR2,
deleteoldrdn IN PLS_INTEGER,
serverctrls IN LDAPCONTROL,
clientctrls IN LDAPCONTROL
)
RETURN PLS_INTEGER;

```

Parameters

Table 12–82 *RENAME_S Function Parameters*

Parameter	Description
ld	This parameter is a handle to an LDAP session returned by a successful call to DBMS_LDAP.init().
dn	This parameter specifies the name of the directory entry to be renamed or moved.
newrdn	This parameter specifies the new RDN.
newparent	This parameter specifies the DN of the new parent.
deleteoldrdn	This parameter specifies whether the old RDN should be retained. If this value is 1, the old RDN is removed.

Table 12–82 (Cont.) RENAME_S Function Parameters

Parameter	Description
serverctrls	Currently not supported.
clientctrls	Currently not supported.

Return Values**Table 12–83 RENAME_S Function Return Values**

Value	Description
PLS_INTEGER	The indication of the success or failure of the operation.

Exceptions**Table 12–84 RENAME_S Function Exceptions**

Exception	Description
invalid_session	Invalid LDAP Session.
invalid_entry_dn	Invalid LDAP DN.
invalid_rdn	Invalid LDAP RDN.
invalid_newparent	Invalid LDAP newparent.
invalid_deleteoldrdn	Invalid LDAP deleteoldrdn.

See Also

DBMS_LDAP.modrdn2_s().

FUNCTION explode_dn

Breaks a DN up into its components.

Syntax

```
FUNCTION explode_dn
(
  dn      IN VARCHAR2,
  notypes IN PLS_INTEGER
)
RETURN STRING_COLLECTION;
```

Parameters**Table 12–85 EXPLODE_DN Function Parameters**

Parameter	Description
dn	This parameter specifies the name of the directory entry to be broken up.
notypes	This parameter specifies whether the attribute tags will be returned. If this value is not 0, no attribute tags are returned.

Return Values

Table 12–86 *EXPLODE_DN Function Return Values*

Value	Description
STRING_COLLECTION	An array of strings. If the DN cannot be broken up, NULL will be returned.

Exceptions

Table 12–87 *EXPLODE_DN Function Exceptions*

Exception	Description
invalid_entry_dn	Invalid LDAP DN.
invalid_notypes	Invalid LDAP notypes value.

See Also

DBMS_LDAP.get_dn().

FUNCTION open_ssl

Establishes an SSL (Secure Sockets Layer) connection over an existing LDAP connection.

Syntax

```
FUNCTION open_ssl
(
  ld          IN SESSION,
  sslwrl      IN VARCHAR2,
  sslwalletpasswd IN VARCHAR2,
  sslauth     IN PLS_INTEGER
)
RETURN PLS_INTEGER;
```

Parameters

Table 12–88 *OPEN_SSL Function Parameters*

Parameter	Description
ld	This parameter is a handle to an LDAP session that is returned by a successful call to DBMS_LDAP.init().
sslwrl	This parameter specifies the wallet location. Required for one-way or two-way SSL connections.
sslwalletpasswd	This parameter specifies the wallet password. Required for one-way or two-way SSL connections.
sslauth	This parameter specifies the SSL Authentication Mode. (1 for no authentication, 2 for one-way authentication required, 3 for two-way authentication).

Return Values**Table 12–89 OPEN_SSL Function Return Values**

Value	Description
PLS_INTEGER	Indicates the success or failure of the operation.

Exceptions**Table 12–90 OPEN_SSL Function Exceptions**

Exception	Description
invalid_session	Invalid LDAP Session.
invalid_ssl_wallet_loc	Invalid LDAP SSL wallet location.
invalid_ssl_wallet_passwd	Invalid LDAP SSL wallet password.
invalid_ssl_auth_mode	Invalid LDAP SSL authentication mode.

Usage Notes

Need to call `DBMS_LDAP.init()` first to acquire a valid ldap session.

See Also

`DBMS_LDAP.init()`.

FUNCTION msgfree

This function frees the chain of messages associated with the message handle returned by synchronous search functions.

Syntax

```
FUNCTION msgfree
(
  res          IN MESSAGE
)
RETURN PLS_INTEGER;
```

Parameters**Table 12–91 MSGFREE Function Parameters**

Parameter	Description
res	The message handle obtained by a call to one of the synchronous search routines.

Return Values

Table 12–92 MSGFREE Return Values

Value	Description
PLS_INTEGER	<p>Indicates the type of the last message in the chain.</p> <p>The function might return any of the following values:</p> <ul style="list-style-type: none"> ■ DBMS_LDAP.LDAP_RES_BIND ■ DBMS_LDAP.LDAP_RES_SEARCH_ENTRY ■ DBMS_LDAP.LDAP_RES_SEARCH_REFERENCE ■ DBMS_LDAP.LDAP_RES_SEARCH_RESULT ■ DBMS_LDAP.LDAP_RES_MODIFY ■ DBMS_LDAP.LDAP_RES_ADD ■ DBMS_LDAP.LDAP_RES_DELETE ■ DBMS_LDAP.LDAP_RES_MODDN ■ DBMS_LDAP.LDAP_RES_COMPARE ■ DBMS_LDAP.LDAP_RES_EXTENDED

Exceptions

`msgfree` raises no exceptions.

See Also

`DBMS_LDAP.search_s()`, `DBMS_LDAP.search_st()`.

FUNCTION ber_free

This function frees the memory associated with a handle to BER_ELEMENT.

Syntax

```
FUNCTION ber_free
(
  ber_elem IN BER_ELEMENT,
  freebuf  IN PLS_INTEGER
)
```

Parameters

Table 12–93 BER_FREE Function Parameters

Parameter	Description
<code>ber_elem</code>	A handle to BER_ELEMENT.
<code>freebuf</code>	<p>The value of this flag should be 0 while the BER_ELEMENT returned from <code>DBMS_LDAP.first_attribute()</code> is being freed. For any other case, the value of this flag should be 1.</p> <p>The default value of this parameter is zero.</p>

Return Values

`ber_free` returns no values.

Exceptions

`ber_free` raises no exceptions.

See Also

DBMS_LDAP.first_attribute(),DBMS_LDAP.next_attribute().

FUNCTION nls_convert_to_utf8

The `nls_convert_to_utf8()` function converts the input string containing database character set data to UTF8 character set data and returns it.

Syntax

```
Function nls_convert_to_utf8
(
  data_local IN VARCHAR2
)
RETURN VARCHAR2;
```

Parameters**Table 12–94 Parameters for nls_convert_to_utf8**

Parameter	Description
data_local	Contains the database character set data.

Return Values**Table 12–95 Return Values for nls_convert_to_utf8**

Value	Description
VARCHAR2	UTF8 character set data string.

Usage Notes

The functions in DBMS_LDAP package expect the input data to be UTF8 character set data if the UTF8_CONVERSION package variable is set to FALSE. The `nls_convert_to_utf8()` function converts database character set data to UTF8 character set data.

If the UTF8_CONVERSION package variable of the DBMS_LDAP package is set to TRUE, functions in the DBMS_LDAP package expect input data to be database character set data.

See Also

DBMS_LDAP.nls_convert_from_utf8(),DBMS_LDAP.nls_get_dbcharset_name().

FUNCTION nls_convert_to_utf8

The `nls_convert_to_utf8()` function converts the input string collection containing database character set data to UTF8 character set data. It then returns the converted data.

Syntax

```
Function nls_convert_to_utf8
(
  data_local IN STRING_COLLECTION
)
RETURN STRING_COLLECTION;
```

Parameters

Table 12–96 *Parameters for nls_convert_to_utf8*

Parameter	Description
data_local	Collection of strings containing database character set data.

Return Values

Table 12–97 *Return Values for nls_convert_to_utf8*

Value	Description
STRING_COLLECTION	Collection of strings containing UTF8 character set data.

Usage Notes

The functions in the DBMS_LDAP package expect the input data to be in the UTF8 character set if the UTF8_CONVERSION package variable is set to FALSE. The `nls_convert_to_utf8()` function converts the input data from the database character set to the UTF8 character set.

If the UTF8_CONVERSION package variable of the DBMS_LDAP package is set to TRUE, functions in the DBMS_LDAP package expect the input data to be in the database character set.

See Also

`DBMS_LDAP.nls_convert_from_utf8()`, `DBMS_LDAP.nls_get_dbcharset_name()`.

FUNCTION `nls_convert_from_utf8`

The `nls_convert_from_utf8()` function converts the input string containing UTF8 character set to database character set data. It then returns this data.

Syntax

```
Function nls_convert_from_utf8
(
  data_utf8 IN VARCHAR2
)
RETURN VARCHAR2;
```

Parameters

Table 12–98 *Parameter for nls_convert_from_utf8*

Parameter	Description
data_utf8	Contains UTF8 character set data.

Return Values

Table 12–99 *Return Value for nls_convert_from_utf8*

Value	Description
VARCHAR2	Data string in the database character set.

Usage Notes

The functions in the `DBMS_LDAP` package return UTF8 character set data if the `UTF8_CONVERSION` package variable is set to `FALSE`. The `nls_convert_from_utf8()` function converts the output data from the UTF8 character set to the database character set.

If the `UTF8_CONVERSION` package variable of the `DBMS_LDAP` package is set to `TRUE`, functions in the `DBMS_LDAP` package return database character set data.

See Also

`DBMS_LDAP.nls_convert_to_utf8()`, `DBMS_LDAP.nls_get_dbcharset_name()`.

FUNCTION `nls_convert_from_utf8`

The `nls_convert_from_utf8()` function converts the input string collection containing UTF8 character set data to database character set data. It then returns this data.

Syntax

```
Function nls_convert_from_utf8
(
  data_utf8 IN STRING_COLLECTION
)
RETURN STRING_COLLECTION;
```

Parameters**Table 12–100** *Parameter for `nls_convert_from_utf8`*

Parameter	Description
<code>data_utf8</code>	Collection of strings containing UTF8 character set data.

Return Values**Table 12–101** *Return Value for `nls_convert_from_utf8`*

Value	Description
<code>VARCHAR2</code>	Collection of strings containing database character set data.

Usage Notes

The functions in the `DBMS_LDAP` package return UTF8 character set data if the `UTF8_CONVERSION` package variable is set to `FALSE`. `nls_convert_from_utf8()` converts the output data from the UTF8 character set to the database character set. If the `UTF8_CONVERSION` package variable of the `DBMS_LDAP` package is set to `TRUE`, functions in the `DBMS_LDAP` package return database character set data.

See Also

`DBMS_LDAP.nls_convert_to_utf8()`, `DBMS_LDAP.nls_get_dbcharset_name()`.

FUNCTION `nls_get_dbcharset_name`

The `nls_get_dbcharset_name()` function returns a string containing the database character set name.

Syntax

Function `nls_get_dbcharset_name`

RETURN VARCHAR2;

Parameters

None.

Return Values

Table 12–102 Return Value for `nls_get_dbcharset_name`

Value	Description
VARCHAR2	String containing the database character set name.

See Also

`DBMS_LDAP.nls_convert_to_utf8()`, `DBMS_LDAP.nls_convert_from_utf8()`.

Java API Reference

The standard Java APIs for Oracle Internet Directory are available as the Java Naming and Directory Interface (JNDI) from Sun Microsystems. The JNDI is found at this link:

<http://java.sun.com/products/jndi>

The Oracle extensions to the standard APIs are found in *Oracle Internet Directory API Reference*.

Sample code for the Java APIs is available at this URL:

http://www.oracle.com/technology/sample_code/

Look for the Oracle Identity Management link under Sample Applications–Oracle Application Server.



DBMS_LDAP_UTL PL/SQL Reference

This chapter contains reference material for the `DBMS_LDAP_UTL` package, which contains Oracle Extension utility functions. The chapter contains these topics:

- [Summary of Subprograms](#)
- [Subprograms](#)
- [Function Return Code Summary](#)
- [Data Type Summary](#)

Note: Sample code for the `DBMS_LDAP_UTL` package is available at this URL:

http://www.oracle.com/technology/sample_code/

Look for the Oracle Identity Management link under Sample Applications–Oracle Application Server.

Summary of Subprograms

Table 14–1 DBMS_LDAP_UTL User-Related Subprograms

Function or Procedure	Purpose
Function <code>authenticate_user</code>	Authenticates a user against an LDAP server.
Function <code>create_user_handle</code>	Creates a user handle.
Function <code>set_user_handle_properties</code>	Associates the given properties to the user handle.
Function <code>get_user_properties</code>	Retrieves user properties from an LDAP server.
Function <code>set_user_properties</code>	Modifies the properties of a user.
Function <code>get_user_extended_properties</code>	Retrieves user extended properties.
Function <code>get_user_dn</code>	Retrieves a user DN.
Function <code>check_group_membership</code>	Checks whether a user is member of a given group.
Function <code>locate_subscriber_for_user</code>	Retrieves the subscriber for the given user.
Function <code>get_group_membership</code>	Retrieves a list of groups of which the user is a member.

Table 14–2 DBMS_LDAP_UTL Group-Related Subprograms

Function or Procedure	Purpose
Function <code>create_group_handle</code>	Creates a group handle.
Function <code>set_group_handle_properties</code>	Associates the given properties with the group handle.
Function <code>get_group_properties</code>	Retrieves group properties from an LDAP server.
Function <code>get_group_dn</code>	Retrieves a group DN.

Table 14–3 DBMS_LDAP_UTL Subscriber-Related Subprograms

Function or Procedure	Purpose
Function <code>create_subscriber_handle</code>	Creates a subscriber handle.
Function <code>get_subscriber_properties</code>	Retrieves subscriber properties from an LDAP server.
Function <code>get_subscriber_dn</code>	Retrieves a subscriber DN.

Table 14–4 DBMS_LDAP_UTL Miscellaneous Subprograms

Function or Procedure	Purpose
Function <code>normalize_dn_with_case</code>	Normalizes the DN string.
Function <code>get_property_names</code>	Retrieves a list of property names in a <code>PROPERTY_SET</code> .
Function <code>get_property_values</code>	Retrieves a list of values for a property name.
Function <code>get_property_values_blob</code>	Retrieves a list of large binary values for a property name.
Procedure <code>property_value_free_blob</code>	Frees the memory associated with <code>BLOB_COLLECTION</code> returned by <code>DBMS_LDAP_UTL.get_property_values_blob()</code> .
Function <code>get_property_values_len</code>	Retrieves a list of binary values for a property name.
Procedure <code>free_propertyset_collection</code>	Frees <code>PROPERTY_SET_COLLECTION</code> .
Function <code>create_mod_propertyset</code>	Creates a <code>MOD_PROPERTY_SET</code> .
Function <code>populate_mod_propertyset</code>	Populates a <code>MOD_PROPERTY_SET</code> structure.
Procedure <code>free_mod_propertyset</code>	Frees a <code>MOD_PROPERTY_SET</code> .
Procedure <code>free_handle</code>	Frees handles.
Function <code>check_interface_version</code>	Checks for support of the interface version.

Subprograms

This section contains the following topics:

- [User-Related Subprograms](#)
- [Group-Related Subprograms](#)
- [Subscriber-Related Subprograms](#)
- [Property-Related Subprograms](#)
- [Miscellaneous Subprograms](#)

User-Related Subprograms

A user is represented by the `DBMS_LDAP_UTL.HANDLE` data type. You can create a user handle by using a DN, GUID, or simple name, along with the appropriate subscriber handle. When a simple name is used, additional information from the root Oracle Context and the subscriber Oracle Context is used to identify the user. This example shows a user handle being created:

```
retval := DBMS_LDAP_UTL.create_user_handle(
user_handle,
DBMS_LDAP_UTL.TYPE_DN,
"cn=user1,cn=users,o=acme,dc=com"
);
```

This user handle must be associated with an appropriate subscriber handle. If, for example, `subscriber_handle` is `o=acme,dc=com`, the subscriber handle can be associated in the following way:

```
retval := DBMS_LDAP_UTL.set_user_handle_properties(
user_handle,
DBMS_LDAP_UTL.SUBSCRIBER_HANDLE,
subscriber_handle
);
```

Common uses of user handles include setting and getting user properties and authenticating the user. Here is a handle that authenticates a user:

```
retval := DBMS_LDAP_UTL.authenticate_user(
my_session
user_handle
DBMS_LDAP_UTL.AUTH_SIMPLE,
"welcome"
NULL
);
```

In this example, the user is authenticated using a clear text password `welcome`.

Here is a handle that retrieves a user's telephone number:

```
--my_attrs is of type DBMS_LDAP.STRING_COLLECTION
my_attrs(1) := 'telephonenumber';
retval := DBMS_LDAP_UTL.get_user_properties(
my_session,
my_attrs,
DBMS_LDAP_UTL.ENTRY_PROPERTIES,
my_pset_coll
);
```

Function `authenticate_user`

The function `authenticate_user()` authenticates the user against Oracle Internet Directory.

Syntax

```
FUNCTION authenticate_user
(
ld IN SESSION,
user_handle IN HANDLE,
auth_type IN PLS_INTEGER,
credentials IN VARCHAR2,
binary_credentials IN RAW
```

```
)
RETURN PLS_INTEGER;
```

Parameters

Table 14–5 *authenticate_user Function Parameters*

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
user_handle	HANDLE	The user handle.
auth_type	PLS_INTEGER	Type of authentication. The only valid value is <code>DBMS_LDAP_UTL.AUTH_SIMPLE</code>
credentials	VARCHAR2	The user credentials.
binary_credentials	RAW	The binary credentials. This parameter is optional. It can be <code>NULL</code> by default.

Return Values

Table 14–6 *authenticate_user Function Return Values*

Value	Description
<code>DBMS_LDAP_UTL.SUCCESS</code>	On a successful completion.
<code>DBMS_LDAP_UTL.PARAM_ERROR</code>	Invalid input parameters.
<code>DBMS_LDAP_UTL.GENERAL_ERROR</code>	Authentication failed.
<code>DBMS_LDAP_UTL.NO_SUCH_USER</code>	User does not exist.
<code>DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES</code>	The user has multiple DN entries.
<code>DBMS_LDAP_UTL.INVALID_SUBSCRIBER_ORCL_CTX</code>	Invalid Subscriber Oracle Context.
<code>DBMS_LDAP_UTL.NO_SUCH_SUBSCRIBER</code>	Subscriber doesn't exist.
<code>DBMS_LDAP_UTL.MULTIPLE_SUBSCRIBER_ENTRIES</code>	The subscriber has multiple DN entries.
<code>DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX</code>	Invalid Root Oracle Context.
<code>DBMS_LDAP_UTL.ACCT_TOTALLY_LOCKED_EXCP</code>	User account is locked.
<code>DBMS_LDAP_UTL.AUTH_PASSWD_CHANGE_WARN</code>	This return value is deprecated.
<code>DBMS_LDAP_UTL.AUTH_FAILURE_EXCP</code>	Authentication failed.
<code>DBMS_LDAP_UTL.PWD_EXPIRED_EXCP</code>	User password has expired.
<code>DBMS_LDAP_UTL.PWD_GRACELOGIN_WARN</code>	Grace login for user.
DBMS_LDAP error codes	Return proper <code>DBMS_LDAP</code> error codes for unconditional failures that occurred when LDAP operations were carried out.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

See Also

`DBMS_LDAP.init()`, `DBMS_LDAP_UTL.create_user_handle()`.

Function create_user_handle

The function `create_user_handle()` creates a user handle.

Syntax

```
FUNCTION create_user_handle
(
  user_hd OUT HANDLE,
  user_type IN PLS_INTEGER,
  user_id IN VARCHAR2,
)
RETURN PLS_INTEGER;
```

Parameters

Table 14–7 CREATE_USER_HANDLE Function Parameters

Parameter Name	Parameter Type	Parameter Description
<code>user_hd</code>	HANDLE	A pointer to a handle to a user.
<code>user_type</code>	PLS_INTEGER	The type of user ID that is passed. Valid values for this argument are as follows: <ul style="list-style-type: none"> ■ <code>DBMS_LDAP_UTL.TYPE_DN</code> ■ <code>DBMS_LDAP_UTL.TYPE_GUID</code> ■ <code>DBMS_LDAP_UTL.TYPE_NICKNAME</code>
<code>user_id</code>	VARCHAR2	The user ID representing the user entry.

Return Values

Table 14–8 CREATE_USER_HANDLE Function Return Values

Value	Description
<code>DBMS_LDAP_UTL.SUCCESS</code>	On a successful completion.
<code>DBMS_LDAP_UTL.PARAM_ERROR</code>	Invalid input parameters.
<code>DBMS_LDAP_UTL.GENERAL_ERROR</code>	Other error.

See Also

`DBMS_LDAP_UTL.get_user_properties()`, `DBMS_LDAP_UTL.set_user_handle_properties()`.

Function set_user_handle_properties

The function `set_user_handle_properties()` configures the user handle properties.

Syntax

```
FUNCTION set_user_handle_properties
(
  user_hd IN HANDLE,
  property_type IN PLS_INTEGER,
  property IN HANDLE
)
RETURN PLS_INTEGER;
```

Parameters

Table 14–9 SET_USER_HANDLE_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
user_hd	HANDLE	A pointer to a handle to a user.
property_type	PLS_INTEGER	The type of property that is passed. Valid values for this argument are as follows: - DBMS_LDAP_UTL.SUBSCRIBER_HANDLE.
property	HANDLE	The property describing the user entry.

Return Values

Table 14–10 SET_USER_HANDLE_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.RESET_HANDLE	When a caller tries to reset the existing handle properties.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

Usage Notes

The subscriber handle does not have to be set in User Handle Properties if the user handle is created with TYPE_DN or TYPE_GUID as the user type.

See Also

DBMS_LDAP_UTL.get_user_properties().

Function get_user_properties

The function get_user_properties() retrieves the user properties.

Syntax

```
FUNCTION get_user_properties
(
  ld IN SESSION,
  user_handle IN HANDLE,
  attrs IN STRING_COLLECTION,
  ptype IN PLS_INTEGER,
  ret_pset_coll OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 14–11 GET_USER_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
user_handle	HANDLE	The user handle.
attrs	STRING_COLLECTION	The list of user attributes to retrieve.

Table 14–11 (Cont.) GET_USER_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
ptype	PLS_INTEGER	Type of properties to return. These are valid values: <ul style="list-style-type: none"> ▪ DBMS_LDAP_UTL.ENTRY_PROPERTIES ▪ DBMS_LDAP_UTL.NICKNAME_PROPERTY
ret-pset_collection	PROPERTY_SET_COLLECTION	User details contained in attributes requested by the caller.

Return Values

Table 14–12 GET_USER_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_SUCH_USER	User does not exist.
DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES	The user has multiple DN entries.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures that occur when LDAP operations are carried out.

Usage Notes

This function requires the following:

- A valid LDAP session handle, which must be obtained from the `DBMS_LDAP.init()` function.
- A valid subscriber handle to be set in the group handle properties if the user type is of `DBMS_LDAP_UTL.TYPE_NICKNAME`.

This function does not identify a `NULL` subscriber handle as a default subscriber. The default subscriber can be obtained from `DBMS_LDAP_UTL.create_subscriber_handle()`, where a `NULL` `subscriber_id` is passed as an argument.

If the group type is either `DBMS_LDAP_UTL.TYPE_GUID` or `DBMS_LDAP_UTL.TYPE_DN`, the subscriber handle need not be set in the user handle properties. If the subscriber handle is set, it is ignored.

See Also

`DBMS_LDAP.init()`, `DBMS_LDAP_UTL.create_user_handle()`.

Function set_user_properties

The function `set_user_properties()` modifies the properties of a user.

Syntax

```
FUNCTION set_user_properties
(
```

```

ld IN SESSION,
user_handle IN HANDLE,
pset_type IN PLS_INTEGER,
mod_pset IN PROPERTY_SET,
mod_op IN PLS_INTEGER
)
RETURN PLS_INTEGER;

```

Parameters

Table 14–13 *SET_USER_PROPERTIES Function Parameters*

Parameter Name	Parameter Type	Description
ld	SESSION	A valid LDAP session handle.
user_handle	HANDLE	The user handle.
pset_type	PLS_INTEGER	The type of property set being modified. A valid value is ENTRY_PROPERTIES.
mod_pset	PROPERTY_SET	Data structure containing modify operations to perform on the property set.
mod_op	PLS_INTEGER	The type of modify operation to be performed on the property set. Here are valid values: <ul style="list-style-type: none"> ■ ADD_PROPERTYSET ■ MODIFY_PROPERTYSET ■ DELETE_PROPERTYSET

Return Values

Table 14–14 *SET_USER_PROPERTIES Function Return Values*

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.NO_SUCH_USER	User does not exist.
DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES	The user has multiple DN entries.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid root Oracle Context.
DBMS_LDAP_UTL.PWD_MIN_LENGTH_ERROR	Password length is less than the minimum required length.
DBMS_LDAP_UTL.PWD_NUMERIC_ERROR	Password must contain numeric characters.
DBMS_LDAP_UTL.PWD_NULL_ERROR	Password cannot be NULL.
DBMS_LDAP_UTL.PWD_INHISTORY_ERROR	Password cannot be the same as the one that is being replaced.
DBMS_LDAP_UTL.PWD_ILLEGALVALUE_ERROR	Password contains illegal characters.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures while carrying out LDAP operations by the LDAP server.

Usage Notes

This function can only be called after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

See Also

DBMS_LDAP.init(), DBMS_LDAP_UTL.get_user_properties().

Function get_user_extended_properties

The function get_user_extended_properties() retrieves user extended properties.

Syntax

```
FUNCTION get_user_extended_properties
(
  ld IN SESSION,
  user_handle IN HANDLE,
  attrs IN STRING_COLLECTION
  ptype IN PLS_INTEGER,
  filter IN VARCHAR2,
  rep_pset_coll OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters**Table 14–15 GET_USER_EXTENDED_PROPERTIES Function Parameters**

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
user_handle	HANDLE	The user handle.
attrs	STRING_COLLECTION	A list of attributes to fetch for the user.
ptype	PLS_INTEGER	The type of properties to return. Here is a valid value: - DBMS_LDAP_UTL.EXTPROPTYPE_RAD
filter	VARCHAR2	An LDAP filter to further refine the user properties returned by the function.
ret_pset_collection	PROPERTY_SET_COLLECTION	The user details containing the attributes requested by the caller.

Return Values**Table 14–16 GET_USER_EXTENDED_PROPERTIES Function Return Values**

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_SUCH_USER	User does not exist.
DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES	The user has multiple DN entries.
USER_PROPERTY_NOT_FOUND	User extended property does not exist.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

Table 14–16 (Cont.) GET_USER_EXTENDED_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures that occur when LDAP operations are carried out.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

See Also

`DBMS_LDAP.init()`, `DBMS_LDAP_UTL.get_user_properties()`.

Function get_user_dn

The function `get_user_dn()` returns the user DN.

Syntax

```
FUNCTION get_user_dn
(
  ld IN SESSION,
  user_handle IN HANDLE,
  dn OUT VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters**Table 14–17 GET_USER_DN Function Parameters**

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
user_handle	HANDLE	The user handle.
dn	VARCHAR2	The user DN.

Return Values**Table 14–18 GET_USER_DN Function Return Values**

Value	Description
<code>DBMS_LDAP_UTL.SUCCESS</code>	On a successful completion.
<code>DBMS_LDAP_UTL.PARAM_ERROR</code>	Invalid input parameters.
<code>DBMS_LDAP_UTL.GENERAL_ERROR</code>	Authentication failed.
<code>DBMS_LDAP_UTL.NO_SUCH_USER</code>	User does not exist.
<code>DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES</code>	The user has multiple DN entries.
<code>DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX</code>	Invalid root Oracle Context.
<code>DBMS_LDAP_UTL.GENERAL_ERROR</code>	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures that occur when LDAP operations are carried out.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

See Also

`DBMS_LDAP.init()`.

Function `check_group_membership`

The function `check_group_membership()` checks whether the user belongs to a group.

Syntax

```
FUNCTION check_group_membership
(
  ld IN SESSION,
  user_handle IN HANDLE,
  group_handle IN HANDLE,
  nested IN PLS_INTEGER
)
RETURN PLS_INTEGER;
```

Parameters

Table 14–19 *CHECK_GROUP_MEMBERSHIP Function Parameters*

Parameter Name	Parameter Type	Parameter Description
<code>ld</code>	SESSION	A valid LDAP session handle.
<code>user_handle</code>	HANDLE	The user handle.
<code>group_handle</code>	HANDLE	The group handle.
<code>nested</code>	PLS_INTEGER	The type of membership the user holds in groups. Here are valid values: <ul style="list-style-type: none"> ▪ <code>DBMS_LDAP_UTL.NESTED_MEMBERSHIP</code> ▪ <code>DBMS_LDAP_UTL.DIRECT_MEMBERSHIP</code>

Return Values

Table 14–20 *CHECK_GROUP_MEMBERSHIP Function Return Values*

Value	Description
<code>DBMS_LDAP_UTL.SUCCESS</code>	If user is a member.
<code>DBMS_LDAP_UTL.PARAM_ERROR</code>	Invalid input parameters.
<code>DBMS_LDAP_UTL.GROUP_MEMBERSHIP</code>	If user is not a member.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

See Also

`DBMS_LDAP.get_group_membership()`.

Function locate_subscriber_for_user

The function `locate_subscriber_for_user()` retrieves the subscriber for the given user and returns a handle to it.

Syntax

```
FUNCTION locate_subscriber_for_user
(
  ld IN SESSION,
  user_handle IN HANDLE,
  subscriber_handle OUT HANDLE
)
RETURN PLS_INTEGER;
```

Parameters

Table 14–21 LOCATE_SUBSCRIBER_FOR_USER Function Parameters

Parameter Name	Parameter Type	Parameter Description
<code>ld</code>	SESSION	A valid LDAP session handle.
<code>user_handle</code>	HANDLE	The user handle.
<code>subscriber_handle</code>	HANDLE	The subscriber handle.

Return Values

Table 14–22 LOCATE SUBSCRIBER FOR USER Function Return Values

Value	Description
<code>DBMS_LDAP_UTL.SUCCESS</code>	On a successful completion.
<code>DBMS_LDAP_UTL.NO_SUCH_SUBSCRIBER</code>	Subscriber doesn't exist.
<code>DBMS_LDAP_UTL.MULTIPLE_SUBSCRIBER_ENTRIES</code>	Multiple number of subscriber DN entries exist in the directory for the given subscriber.
<code>DBMS_LDAP_UTL.NO_SUCH_USER</code>	User doesn't exist.
<code>DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES</code>	Multiple number of user DN entries exist in the directory for the given user.
<code>DBMS_LDAP_UTL.SUBSCRIBER_NOT_FOUND</code>	Unable to locate subscriber for the given user.
<code>DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX</code>	Invalid Root Oracle Context.
<code>DBMS_LDAP_UTL.ACCT_TOTALLY_LOCKED_EXCP</code>	User account is locked.
<code>DBMS_LDAP_UTL.GENERAL_ERROR</code>	Other error.
DBMS_LDAP error codes	Return proper <code>DBMS_LDAP</code> error codes for unconditional failures while carrying out LDAP operations by the LDAP server.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

See Also

`DBMS_LDAP.init()`, `DBMS_LDAP_UTL.create_user_handle()`.

Function `get_group_membership`

The function `get_group_membership()` returns the list of groups to which the user is a member.

Syntax

```
FUNCTION get_group_membership
(
  user_handle IN HANDLE,
  nested IN PLS_INTEGER,
  attr_list IN STRING_COLLECTION,
  ret_groups OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 14–23 *GET_GROUP_MEMBERSHIP Function Parameters*

Parameter Name	Parameter Type	Parameter Description
<code>ld</code>	SESSION	A valid LDAP session handle.
<code>user_handle</code>	HANDLE	The user handle.
<code>nested</code>	PLS_INTEGER	The type of membership the user holds in groups. Here are valid values: <ul style="list-style-type: none"> ▪ <code>DBMS_LDAP_UTL.NESTED_MEMBERSHIP</code> ▪ <code>DBMS_LDAP_UTL.DIRECT_MEMBERSHIP</code>
<code>attr_list</code>	STRING_COLLECTION	A list of attributes to be returned.
<code>ret_groups</code>	PROPERTY_SET_COLLECTION	A pointer to a pointer to an array of group entries.

Return Values

Table 14–24 *GET_GROUP_MEMBERSHIP Function Return Values*

Value	Description
<code>DBMS_LDAP_UTL.SUCCESS</code>	On a successful completion.
<code>DBMS_LDAP_UTL.PARAM_ERROR</code>	Invalid input parameters.
<code>DBMS_LDAP_UTL.GENERAL_ERROR</code>	Other error.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

See Also

`DBMS_LDAP.init()`.

Group-Related Subprograms

A group is represented using by using the `DBMS_LDAP_UTL.HANDLE` data type. A group handle represents a valid group entry. You can create a group handle by using a DN, GUID or a simple name, along with the appropriate subscriber handle. When a

simple name is used, additional information from the Root Oracle Context and the Subscriber Oracle Context is used to identify the group. Here is an example of a group handle creation:

```
retval := DBMS_LDAP_UTL.create_group_handle(  
group_handle,  
DBMS_LDAP_UTL.TYPE_DN,  
"cn=group1,cn=Groups,o=acme,dc=com"  
);
```

This group handle has to be associated with an appropriate subscriber handle. For example, given a subscriber handle: *subscriber_handle* representing o=acme,dc=com, the subscriber handle can be associated in the following way:

```
retval := DBMS_LDAP_UTL.set_group_handle_properties(  
group_handle,  
DBMS_LDAP_UTL.SUBSCRIBER_HANDLE,  
subscriber_handle  
);
```

A sample use of group handle is getting group properties. Here is an example:

```
my_attrs is of type DBMS_LDAP.STRING_COLLECTION  
my_attrs(1) := 'uniquemember';  
retval := DBMS_LDAP_UTL.get_group_properties(  
my_session,  
my_attrs,  
DBMS_LDAP_UTL.ENTRY_PROPERTIES,  
my_pset_coll  
);
```

The group-related subprograms also support membership-related functionality. Given a user handle, you can find out if it is a direct or a nested member of a group by using the `DBMS_LDAP_UTL.check_group_membership()` function. Here is an example:

```
retval := DBMS_LDAP_UTL.check_group_membership(  
session,  
user_handle,  
group_handle,  
DBMS_LDAP_UTL.DIRECT_MEMBERSHIP
```

You can also obtain a list of groups that a particular group belongs to, using the `DBMS_LDAP_UTL.get_group_membership()` function. For example:

```
my_attrs is of type DBMS_LDAP.STRING_COLLECTION  
my_attrs(1) := 'cn';  
retval := DBMS_LDAP_UTL.get_group_membership(  
my_session,  
user_handle,  
DBMS_LDAP_UTL.DIRECT_MEMBERSHIP,  
my_attrs  
my_pset_coll  
);
```

Function `create_group_handle`

The function `create_group_handle()` creates a group handle.

Syntax

```
FUNCTION create_group_handle  
(
```

```

group_hd OUT HANDLE,
group_type IN PLS_INTEGER,
group_id IN VARCHAR2
)
RETURN PLS_INTEGER;

```

Parameters

Table 14–25 *CREATE_GROUP_HANDLE Function Parameters*

Parameter Name	Parameter Type	Parameter Description
group_hd	HANDLE	A pointer to a handle to a group.
group_type	PLS_INTEGER	The type of group ID that is passed. Valid values for this argument are as follows: <ul style="list-style-type: none"> ■ DBMS_LDAP_UTL.TYPE_DN ■ DBMS_LDAP_UTL.TYPE_GUID ■ DBMS_LDAP_UTL.TYPE_NICKNAME
group_id	VARCHAR2	The group ID representing the group entry.

Return Values

Table 14–26 *CREATE_GROUP_HANDLE Function Return Values*

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

See Also

DBMS_LDAP_UTL.get_group_properties(), DBMS_LDAP_UTL.set_group_handle_properties().

Function set_group_handle_properties

The function set_group_handle_properties() configures the group handle properties.

Syntax

```

FUNCTION set_group_handle_properties
(
group_hd IN HANDLE,
property_type IN PLS_INTEGER,
property IN HANDLE
)
RETURN PLS_INTEGER;

```

Parameters

Table 14–27 *SET_GROUP_HANDLE_PROPERTIES Function Parameters*

Parameter Name	Parameter Type	Parameter Description
group_hd	HANDLE	A pointer to the handle to the group.

Table 14–27 (Cont.) SET_GROUP_HANDLE_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
property_type	PLS_INTEGER	The type of property that is passed. Valid values for this argument are as follows: DBMS_LDAP_UTL.GROUP_HANDLE
property	HANDLE	The property describing the group entry.

Return Values

Table 14–28 SET_GROUP_HANDLE_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.RESET_HANDLE	When a caller tries to reset the existing handle properties.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

Usage Notes

The subscriber handle doesn't need to be set in Group Handle Properties if the group handle is created with TYPE_DN or TYPE_GUID as the group type.

See Also

DBMS_LDAP_UTL.get_group_properties().

Function get_group_properties

The function `get_group_properties()` retrieves the group properties.

Syntax

```
FUNCTION get_group_properties
(
  ld IN SESSION,
  group_handle IN HANDLE,
  attrs IN STRING_COLLECTION,
  ptype IN PLS_INTEGER,
  ret_pset_coll OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 14–29 GET_GROUP_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
group_handle	HANDLE	The group handle.
attrs	STRING_COLLECTION	A list of attributes that must be fetched for the group.
ptype	PLS_INTEGER	The type of properties to be returned. The valid value is DBMS_LDAP_UTL.ENTRY_PROPERTIES

Table 14–29 (Cont.) GET_GROUP_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
ret_pset_coll	PROPERTY_SET_COLLECTION	The group details containing the attributes requested by the caller.

Return Values

Table 14–30 GET_GROUP_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_SUCH_GROUP	Group doesn't exist.
DBMS_LDAP_UTL.MULTIPLE_GROUP_ENTRIES	Multiple number of group DN entries exist in the directory for the given group.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid Root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures while carrying out LDAP operations by the LDAP server.

Usage Notes

This function requires the following:

- A valid LDAP session handle which must be obtained from the `DBMS_LDAP.init()` function.
- A valid subscriber handle to be set in the group handle properties if the group type is of: `DBMS_LDAP_UTL.TYPE_NICKNAME`.

This function does not identify a `NULL` subscriber handle as a default subscriber. The default subscriber can be obtained from `DBMS_LDAP_UTL.create_subscriber_handle()`, where a `NULL` `subscriber_id` is passed as an argument.

If the group type is either `DBMS_LDAP_UTL.TYPE_GUID` or `DBMS_LDAP_UTL.TYPE_DN`, the subscriber handle does not have to be set in the group handle properties. If the subscriber handle is set, it is ignored.

See Also

`DBMS_LDAP.init()`, `DBMS_LDAP_UTL.create_group_handle()`.

Function get_group_dn

The function `get_group_dn()` returns the group DN.

Syntax

```
FUNCTION get_group_dn
(
  ld IN SESSION,
  group_handle IN HANDLE
  dn OUT VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 14–31 *GET_GROUP_DN Function Parameters*

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
group_handle	HANDLE	The group handle.
dn	VARCHAR2	The group DN.

Return Values

Table 14–32 *GET_GROUP_DN Function Return Values*

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_SUCH_GROUP	Group doesn't exist.
DBMS_LDAP_UTL.MULTIPLE_GROUP_ENTRIES	Multiple number of group DN entries exist in the directory for the given group.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid Root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures that are encountered when LDAP operations are carried out.

Usage Notes

This function can only be called after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

See Also

`DBMS_LDAP.init()`.

Subscriber-Related Subprograms

A subscriber is represented by using `dbms_ldap_util.handle` data type. You can create a subscriber handle by using a DN, GUID or simple name. When a simple name is used, additional information from the root Oracle Context is used to identify the subscriber. This example shows a subscriber handle being created:

```
retval := DBMS_LDAP_UTL.create_subscriber_handle(
subscriber_handle,
DBMS_LDAP_UTL.TYPE_DN,
"o=acme,dc=com"
);
```

`subscriber_handle` is created by its DN: `o=oracle,dc=com`.

Getting subscriber properties is one common use of a subscriber handle. Here is an example:

```
my_attrs is of type DBMS_LDAP.STRING_COLLECTION
my_attrs(1) := 'orclguid';
```

```

        retval := DBMS_LDAP_UTL.get_subscriber_properties(
my_session,
my_attrs,
DBMS_LDAP_UTL.ENTRY_PROPERTIES,
my_pset_coll
);

```

Function `create_subscriber_handle`

The function `create_subscriber_handle()` creates a subscriber handle.

Syntax

```

FUNCTION create_subscriber_handle
(
subscriber_hd OUT HANDLE,
subscriber_type IN PLS_INTEGER,
subscriber_id IN VARCHAR2
)
RETURN PLS_INTEGER;

```

Parameters

Table 14–33 *CREATE_SUBSCRIBER_HANDLE Function Parameters*

Parameter Name	Parameter Type	Parameter Description
<code>subscriber_hd</code>	HANDLE	A pointer to a handle to a subscriber.
<code>subscriber_type</code>	PLS_INTEGER	The type of subscriber ID that is passed. Valid values for this argument are: <ul style="list-style-type: none"> ▪ <code>DBMS_LDAP_UTL.TYPE_DN</code> ▪ <code>DBMS_LDAP_UTL.TYPE_GUID</code> ▪ <code>DBMS_LDAP_UTL.TYPE_NICKNAME</code> ▪ <code>DBMS_LDAP_UTL.TYPE_DEFAULT</code>
<code>subscriber_id</code>	VARCHAR2	The subscriber ID representing the subscriber entry. This can be NULL if <code>subscriber_type</code> is <code>DBMS_LDAP_UTL.TYPE_DEFAULT</code> . In this case, the default subscriber is retrieved from the root Oracle Context.

Return Values

Table 14–34 *CREATE_SUBSCRIBER_HANDLE Function Return Values*

Value	Description
<code>DBMS_LDAP_UTL.SUCCESS</code>	On a successful completion.
<code>DBMS_LDAP_UTL.PARAM_ERROR</code>	Invalid input parameters.
<code>DBMS_LDAP_UTL.GENERAL_ERROR</code>	Other error.

See Also

`DBMS_LDAP_UTL.get_subscriber_properties()`.

Function `get_subscriber_properties`

The function `get_subscriber_properties()` retrieves the subscriber properties for the given subscriber handle.

Syntax

```

FUNCTION get_subscriber_properties
(
  ld IN SESSION,
  subscriber_handle IN HANDLE,
  attrs IN STRING_COLLECTION,
  ptype IN PLS_INTEGER,
  ret_pset_coll OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;

```

Parameters**Table 14–35** *GET_SUBSCRIBER_PROPERTIES Function Parameters*

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
subscriber_handle	HANDLE	The subscriber handle.
attrs	STRING_COLLECTION	A list of attributes that must be retrieved for the subscriber.
ptype	PLS_INTEGER	Properties of the subscriber's Oracle Context to return. These are valid values: <ul style="list-style-type: none"> ■ DBMS_LDAP_UTL.ENTRY_PROPERTIES ■ DBMS_LDAP_UTL.COMMON_PROPERTIES
ret_pset_coll	PROPERTY_SET_COLLECTION	The subscriber details containing the attributes requested by the caller.

Return Values**Table 14–36** *GET_SUBSCRIBER_PROPERTIES Function Return Values*

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_SUCH_SUBSCRIBER	Subscriber doesn't exist.
DBMS_LDAP_UTL.MULTIPLE_SUBSCRIBER_ENTRIES	Subscriber has a multiple number of DN entries.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures encountered while LDAP operations are carried out.

Usage Notes

This function can only be called after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

See Also

DBMS_LDAP.init(), DBMS_LDAP_UTL.create_subscriber_handle().

Function get_subscriber_dn

The function get_subscriber_dn() returns the subscriber DN.

Syntax

```
FUNCTION get_subscriber_dn
(
  ld IN SESSION,
  subscriber_handle IN HANDLE,
  dn OUT VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters**Table 14–37 GET_SUBSCRIBER_DN Function Parameters**

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
subscriber_handle	HANDLE	The subscriber handle.
dn	VARCHAR2	The subscriber DN.

Return Values**Table 14–38 GET_SUBSCRIBER_DN Function Return Values**

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_SUCH_SUBSCRIBER	Subscriber doesn't exist.
DBMS_LDAP_UTL.MULTIPLE_SUBSCRIBER_ENTRIES	Multiple number of subscriber DN entries exist in the directory for the given subscriber.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures encountered when LDAP operations are carried out.

Usage Notes

This function can only be called after a valid LDAP session is obtained from a call to DBMS_LDAP.init().

See Also

DBMS_LDAP.init().

Function `get_subscriber_ext_properties`

The function `get_subscriber_ext_properties()` retrieves the subscriber extended properties. Currently this can be used to retrieve the subscriber-wide default Resource Access Descriptors.

Syntax

```
FUNCTION get_subscriber_ext_properties
(
  ld IN SESSION,
  subscriber_handle IN HANDLE,
  attrs IN STRING_COLLECTION,
  ptype IN PLS_INTEGER,
  filter IN VARCHAR2,
  rep_pset_coll OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 14–39 *GET_SUBSCRIBER_EXT_PROPERTIES Function Parameters*

Parameter Name	Parameter Type	Parameter Description
<code>ld</code>	SESSION	A valid LDAP session handle.
<code>subscriber_handle</code>	HANDLE	The subscriber handle.
<code>attrs</code>	STRING_COLLECTION	A list of subscriber attributes to retrieve.
<code>ptype</code>	PLS_INTEGER	The type of properties to return. A valid value is <code>DBMS_LDAP_UTL.DEFAULT_RAD_PROPERTIES</code> .
<code>filter</code>	VARCHAR2	An LDAP filter to further refine the subscriber properties returned by the function.
<code>ret_pset_collection</code>	PROPERTY_SET_COLLECTION	The subscriber details containing the attributes requested by the caller.

Return Values

Table 14–40 *GET_USER_EXTENDED_PROPERTIES Function Return Values*

Value	Description
<code>DBMS_LDAP_UTL.SUCCESS</code>	On a successful completion.
<code>DBMS_LDAP_UTL.PARAM_ERROR</code>	Invalid input parameters.
<code>DBMS_LDAP_UTL.NO_SUCH_USER</code>	User does not exist.
<code>DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX</code>	Invalid root Oracle Context.
<code>DBMS_LDAP_UTL.GENERAL_ERROR</code>	Other error.
<code>DBMS_LDAP error codes</code>	Return proper <code>DBMS_LDAP</code> error codes for unconditional failures encountered when LDAP operations are carried out.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

See Also `DBMS_LDAP.init()`, `DBMS_LDAP_UTL.get_subscriber_properties()`.

Property-Related Subprograms

Many of the user-related, subscriber-related, and group-related subprograms return `DBMS_LDAP_UTL.PROPERTY_SET_COLLECTION`, which is a collection of one or more LDAP entries representing results. Each of these entries is represented by a `DBMS_LDAP_UTL.PROPERTY_SET`. A `PROPERTY_SET` may contain attributes—that is, properties—and its values. Here is an example that illustrates the retrieval of properties from `DBMS_LDAP_UTL.PROPERTY_SET_COLLECTION`:

```
my_attrs is of type DBMS_LDAP.STRING_COLLECTION
my_attrs(1) := 'cn';

retval := DBMS_LDAP_UTL.get_group_membership(
my_session,
user_handle,
DBMS_LDAP_UTL.DIRECT_MEMBERSHIP,
my_attrs,
my_pset_coll
);

IF my_pset_coll.count > 0 THEN
  FOR i in my_pset_coll.first .. my_pset_coll.last LOOP
    -- my_property_names is of type DBMS_LDAP.STRING_COLLECTION
    retval := DBMS_LDAP_UTL.get_property_names(
pset_coll(i),
property_names
  IF my_property_names.count > 0 THEN
    FOR j in my_property_names.first .. my_property_names.last LOOP
      retval := DBMS_LDAP_UTL.get_property_values(
pset_coll(i),
property_names(j),
property_values
      if my_property_values.COUNT > 0 then
        FOR k in my_property_values.FIRST..my_property_values.LAST LOOP
          DBMS_OUTPUT.PUT_LINE(my_property_names(j) || ':' || my_
property_values(k));
        END LOOP; -- For each value
      else
        DBMS_OUTPUT.PUT_LINE('NO VALUES FOR' || my_property_names(j));
      end if;
    END LOOP; -- For each property name
  END IF; -- IF my_property_names.count > 0
END LOOP; -- For each propertyset
END IF; -- If my_pset_coll.count > 0
```

`user_handle` is a user handle. `my_pset_coll` contains all the nested groups that `user_handle` belongs to. The code loops through the resulting entries and prints out the `cn` of each entry.

Miscellaneous Subprograms

The miscellaneous subprograms in the `DBMS_LDAP_UTL` package perform a variety of different functions.

Function `normalize_dn_with_case`

The function `normalize_dn_with_case()` removes unnecessary white space characters from a DN and converts all characters to lower case based on a flag.

Syntax

```
FUNCTION normalize_dn_with_case
(
  dn IN VARCHAR2,
  lower_case IN PLS_INTEGER,
  norm_dn OUT VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 14–41 *NORMALIZE_DN_WITH_CASE Function Parameters*

Parameter Name	Parameter Type	Parameter Description
<code>dn</code>	<code>VARCHAR2</code>	The DN.
<code>lower_case</code>	<code>PLS_INTEGER</code>	If set to 1: The normalized DN returns in lower case. If set to 0: The case is preserved in the normalized DN string.
<code>norm_dn</code>	<code>VARCHAR2</code>	The normalized DN.

Return Values

Table 14–42 *NORMALIZE_DN_WITH_CASE Function Return Values*

Value	Description
<code>DBMS_LDAP_UTL.SUCCESS</code>	On a successful completion.
<code>DBMS_LDAP_UTL.PARAM_ERROR</code>	Invalid input parameters.
<code>DBMS_LDAP_UTL.GENERAL_ERROR</code>	On failure.

Usage Notes

This function can be used while comparing two DNs.

Function `get_property_names`

The function `get_property_names()` retrieves the list of property names in the property set.

Syntax

```
FUNCTION get_property_names
(
  pset IN PROPERTY_SET,
  property_names OUT STRING_COLLECTION
)
RETURN PLS_INTEGER;
```


Parameters

Table 14–43 *GET_PROPERTY_NAMES Function Parameters*

Parameter Name	Parameter Type	Parameter Description
pset	PROPERTY_SET	The property set in the property set collection returned from any of the following functions: <ul style="list-style-type: none"> ▪ DBMS_LDAP_UTL.get_group_membership() ▪ DBMS_LDAP_UTL.get_subscriber_properties() ▪ DBMS_LDAP_UTL.get_user_properties() ▪ DBMS_LDAP_UTL.get_group_properties()
property_names	STRING_COLLECTION	A list of property names associated with the property set.

Return Values

Table 14–44 *GET_PROPERTY_NAMES Function Return Values*

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	On error.

See Also

DBMS_LDAP_UTL.get_property_values().

Function get_property_values

The function `get_property_values()` retrieves the property values (the strings) for a given property name and property.

Syntax

```
FUNCTION get_property_values
(
  pset IN PROPERTY_SET,
  property_name IN VARCHAR2,
  property_values OUT STRING_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 14–45 *GET_PROPERTY_VALUES Function Parameters*

Parameter Name	Parameter Type	Parameter Description
property_name	VARCHAR2	The property name.

Table 14–45 (Cont.) GET_PROPERTY_VALUES Function Parameters

Parameter Name	Parameter Type	Parameter Description
pset	PROPERTY_SET	The property set in the property set collection obtained from any of the following function returns: <ul style="list-style-type: none"> ■ DBMS_LDAP_UTL.get_group_membership() ■ DBMS_LDAP_UTL.get_subscriber_properties() ■ DBMS_LDAP_UTL.get_user_properties() ■ DBMS_LDAP_UTL.get_group_properties()
property_values	STRING_COLLECTION	A list of property values (strings).

Return Values**Table 14–46 GET_PROPERTY_VALUES Function Return Values**

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	On failure.

See Also

DBMS_LDAP_UTL.get_property_values_len().

Function get_property_values_len

The function `get_property_values_len()` retrieves the binary property values for a given property name and property.

Syntax

```
FUNCTION get_property_values_len
(
  pset IN PROPERTY_SET,
  property_name IN VARCHAR2,
  auth_type IN PLS_INTEGER,
  property_values OUT BINVAL_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters**Table 14–47 GET_PROPERTY_VALUES_LEN Function Parameters**

Parameter Name	Parameter Type	Parameter Description
property_name	VARCHAR2	A property name.

Table 14–47 (Cont.) GET_PROPERTY_VALUES_LEN Function Parameters

Parameter Name	Parameter Type	Parameter Description
pset	PROPERTY_SET	The property set in the property set collection obtained from any of the following function returns: <ul style="list-style-type: none"> ■ DBMS_LDAP_UTL.get_group_membership() ■ DBMS_LDAP_UTL.get_subscriber_properties() ■ DBMS_LDAP_UTL.get_user_properties() ■ DBMS_LDAP_UTL.get_group_properties()
property_values	BINVAL_COLLECTION	A list of binary property values.

Return Values**Table 14–48 GET_PROPERTY_VALUES_LEN Function Return Values**

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	On failure.

See Also

DBMS_LDAP_UTL.get_property_values().

Procedure free_propertyset_collection

The procedure `free_propertyset_collection()` frees the memory associated with property set collection.

Syntax

```
PROCEDURE free_propertyset_collection
(
  pset_collection IN OUT PROPERTY_SET_COLLECTION
);
```

Parameters

Table 14–49 *FREE_PROPERTYSET_COLLECTION Procedure Parameters*

Parameter Name	Parameter Type	Parameter Description
pset_collection	PROPERTY_SET_COLLECTION	<p>The property set collection returned from one of the following functions:</p> <ul style="list-style-type: none"> ▪ DBMS_LDAP_UTL.get_group_membership() ▪ DBMS_LDAP_UTL.get_subscriber_properties() ▪ DBMS_LDAP_UTL.get_user_properties() ▪ DBMS_LDAP_UTL.get_group_properties()

See Also

DBMS_LDAP_UTL.get_group_membership(), DBMS_LDAP_UTL.get_subscriber_properties(), DBMS_LDAP_UTL.get_user_properties(), DBMS_LDAP_UTL.get_group_properties().

Function create_mod_propertyset

The function create_mod_propertyset() creates a MOD_PROPERTY_SET data structure.

Syntax

```
FUNCTION create_mod_propertyset
(
  pset_type IN PLS_INTEGER,
  pset_name IN VARCHAR2,
  mod_pset OUT MOD_PROPERTY_SET
)
RETURN PLS_INTEGER;
```

Parameters

Table 14–50 *CREATE_MOD_PROPERTYSET Function Parameters*

Parameter Name	Parameter Type	Parameter Description
pset_type	PLS_INTEGER	The type of property set being modified. Here is a valid value: ENTRY_PROPERTIES
pset_name	VARCHAR2	The name of the property set. This can be NULL if ENTRY_PROPERTIES are being modified.
mod_pset	MOD_PROPERTY_SET	The data structure to contain modify operations to be performed on the property set.

Return Values

Table 14–51 *CREATE_MOD_PROPERTYSET Function Return Values*

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

See Also

DBMS_LDAP_UTL.populate_mod_propertyset().

Function populate_mod_propertyset

The function `populate_mod_propertyset()` populates the `MOD_PROPERTY_SET` data structure.

Syntax

```
FUNCTION populate_mod_propertyset
(
  mod_pset IN MOD_PROPERTY_SET,
  property_mod_op IN PLS_INTEGER,
  property_name IN VARCHAR2,
  property_values IN STRING_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters**Table 14–52** POPULATE_MOD_PROPERTYSET Function Parameters

Parameter Name	Parameter Type	Parameter Description
<code>mod_pset</code>	<code>MOD_PROPERTY_SET</code>	Mod-PropertySet data structure.
<code>property_mod_op</code>	<code>PLS_INTEGER</code>	The type of modify operation to perform on a property. These are valid values: <ul style="list-style-type: none"> ■ <code>ADD_PROPERTY</code> ■ <code>REPLACE_PROPERTY</code> ■ <code>DELETE_PROPERTY</code>
<code>property_name</code>	<code>VARCHAR2</code>	The name of the property
<code>property_values</code>	<code>STRING_COLLECTION</code>	Values associated with the property.

Return Values**Table 14–53** POPULATE_MOD_PROPERTYSET Function Return Values

Value	Description
<code>DBMS_LDAP_UTL.SUCCESS</code>	On a successful completion.
<code>DBMS_LDAP_UTL.GENERAL_ERROR</code>	Authentication failed.
<code>DBMS_LDAP_UTL.PWD_GRACELOGIN_WARN</code>	Grace login for user.

See Also

DBMS_LDAP_UTL.create_mod_propertyset().

Procedure free_mod_propertyset

The procedure `free_mod_propertyset()` frees the `MOD_PROPERTY_SET` data structure.

Syntax

```
PROCEDURE free_mod_propertyset
(
  mod_pset IN MOD_PROPERTY_SET
```

```
);
```

Parameters

Table 14–54 *FREE_MOD_PROPERTYSET Procedure Parameters*

Parameter Name	Parameter Type	Parameter Description
mod_pset	PROPERTY_SET	Mod_PropertySet data structure.

See Also

DBMS_LDAP_UTL.create_mod_propertyset().

Procedure free_handle

The procedure free_handle() frees the memory associated with the handle.

Syntax

```
PROCEDURE free_handle
(
  handle IN OUT HANDLE
);
```

Parameters

Table 14–55 *FREE_HANDLE Procedure Parameters*

Parameter Name	Parameter Type	Parameter Description
handle	HANDLE	A pointer to a handle.

See Also

DBMS_LDAP_UTL.create_user_handle(), DBMS_LDAP_UTL.create_subscriber_handle(), DBMS_LDAP_UTL.create_group_handle().

Function check_interface_version

The function check_interface_version() checks the interface version.

Syntax

```
FUNCTION check_interface_version
(
  interface_version IN VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 14–56 *CHECK_INTERFACE_VERSION Function Parameters*

Parameter Name	Parameter Type	Parameter Description
interface_version	VARCHAR2	Version of the interface.

Return Values

Table 14–57 *CHECK_VERSION_INTERFACE Function Return Values*

Value	Description
DBMS_LDAP_UTL.SUCCESS	Interface version is supported.
DBMS_LDAP_UTL.GENERAL_ERROR	Interface version is not supported.

Function get_property_values_blob

The function `get_property_values_blob()` retrieves large binary property values for a given property name and property.

Syntax

```
FUNCTION get_property_values_blob
(
  pset IN PROPERTY_SET,
  property_name IN VARCHAR2,
  auth_type IN PLS_INTEGER,
  property_values OUT BLOB_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 14–58 *GET_PROPERTY_VALUES_BLOB Function Parameters*

Parameters	Parameter Type	Description
property_name	VARCHAR2	A property name.
pset	PROPERTY_SET	The property set in the property set collection obtained from any of the following function returns: <ul style="list-style-type: none"> ▪ DBMS_LDAP_UTL.get_group_membership() ▪ DBMS_LDAP_UTL.get_subscriber_properties() ▪ DBMS_LDAP_UTL.get_user_properties() ▪ DBMS_LDAP_UTL.get_group_properties()
property_values	BLOB_COLLECTION	A list of binary property values.

Return Values

Table 14–59 *GET_PROPERTY_VALUES_BLOB Return Values*

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	On failure.

See Also

DBMS_LDAP_UTL.get_property_values().

Procedure `property_value_free_blob`

Frees the memory associated with `BLOB_COLLECTION` returned by `DBMS_LDAP.get_property_values_blob()`.

Syntax

```
Syntax
PROCEDURE property_value_free_blob
(
vals IN OUT DBMS_LDAP.BLOB_COLLECTION
);
```

Parameters**Table 14–60** *PROPERTY_VALUE_FREE_BLOB Function Parameters*

Parameter	Description
<code>vals</code>	The collection of large binary values returned by <code>DBMS_LDAP.get_property_values_blob()</code> .

See Also

`DBMS_LDAP.get_property_values_blob()`.

Function Return Code Summary

The `DBMS_LDAP_UTL` functions can return the values in the following table

Table 14–61 *Function Return Codes*

Name	Return Code	Description
<code>SUCCESS</code>	0	Operation successful.
<code>GENERAL_ERROR</code>	-1	This error code is returned on failure conditions other than those conditions listed here.
<code>PARAM_ERROR</code>	-2	Returned by all functions when an invalid input parameter is encountered.
<code>NO_GROUP_MEMBERSHIP</code>	-3	Returned by user-related functions and group functions when the user is not a member of a group.
<code>NO_SUCH_SUBSCRIBER</code>	-4	Returned by subscriber-related functions when the subscriber does not exist in the directory.
<code>NO_SUCH_USER</code>	-5	Returned by user-related functions when the user does not exist in the directory.
<code>NO_ROOT_ORCL_CTX</code>	-6	Returned by most functions when the root oracle context does not exist in the directory.
<code>MULTIPLE_SUBSCRIBER_ENTRIES</code>	-7	Returned by subscriber-related functions when multiple subscriber entries are found for the given subscriber nickname.
<code>INVALID_ROOT_ORCL_CTX</code>	-8	Root Oracle Context does not contain all the required information needed by the function.
<code>NO_SUBSCRIBER_ORCL_CTX</code>	-9	Oracle Context does not exist for the subscriber.
<code>INVALID_SUBSCRIBER_ORCL_CTX</code>	-10	Oracle Context for the subscriber is invalid.
<code>MULTIPLE_USER_ENTRIES</code>	-11	Returned by user-related functions when multiple user entries exist for the given user nickname.

Table 14–61 (Cont.) Function Return Codes

Name	Return Code	Description
NO_SUCH_GROUP	-12	Returned by group related functions when a group does not exist in the directory.
MULTIPLE_GROUP_ENTRIES	-13	Multiple group entries exist for the given group nickname in the directory.
ACCT_TOTALLY_LOCKED_EXCEPTION	-14	Returned by DBMS_LDAP_UTL.authenticate_user() function when a user account is locked. This error is based on the password policy set in the subscriber oracle context.
AUTH_PASSWD_CHANGE_WARN	-15	This return code is deprecated.
AUTH_FAILURE_EXCEPTION	-16	Returned by DBMS_LDAP_UTL.authenticate_user() function when user authentication fails.
PWD_EXPIRED_EXCEPTION	-17	Returned by DBMS_LDAP_UTL.authenticate_user() function when the user password has expired. This is a password policy error.
RESET_HANDLE	-18	Returned when entity handle properties are being reset by the caller.
SUBSCRIBER_NOT_FOUND	-19	Returned by DBMS_LDAP-UTL.locate_subscriber_for_user() function when it is unable to locate the subscriber.
PWD_EXPIRE_WARN	-20	Returned by DBMS_LDAP_UTL.authenticate_user() function when the user password is about to expire. This is a password policy error.
PWD_MINLENGTH_ERROR	-21	Returned by DBMS_LDAP_UTL.set_user_properties() function while changing the user password and the new user password is less than the minimum required length. This is a password policy error.
PWD_NUMERIC_ERROR	-22	Returned by DBMS_LDAP_UTL.set_user_properties() function while changing the user password and the new user password does not contain at least one numeric character. This is a password policy error.
PWD_NULL_ERROR	-23	Returned by DBMS_LDAP_UTL.set_user_properties() function while changing the user password and the new user password is an empty password. This is a password policy error.
PWD_INHISTORY_ERROR	-24	Returned by DBMS_LDAP_UTL.set_user_properties() function while changing the user password and the new user password is the same as the previous password. This is a password policy error.
PWD_ILLEGALVALUE_ERROR	-25	Returned by DBMS_LDAP_UTL.set_user_properties() function while changing the user password and the new user password has an illegal character. This is a password policy error.
PWD_GRACELOGIN_WARN	-26	Returned by DBMS_LDAP_UTL.authenticate_user() function to indicate that the user password has expired and the user has been given a grace login. This is a password policy error.

Table 14–61 (Cont.) Function Return Codes

Name	Return Code	Description
PWD_MUSTCHANGE_ERROR	-27	Returned by <code>DBMS_LDAP_UTL.authenticate_user()</code> function when user password needs to be changed. This is a password policy error.
USER_ACCT_DISABLED_ERROR	-29	Returned by <code>DBMS_LDAP_UTL.authenticate_user()</code> function when user account has been disabled. This is a password policy error.
PROPERTY_NOT_FOUND	-30	Returned by user-related functions while searching for a user property in the directory.

Data Type Summary

The `DBMS_LDAP_UTL` package uses the data types in the following table

Table 14–62 DBMS_LDAP_UTL Data Types

Data Type	Purpose
HANDLE	Used to hold the entity.
PROPERTY_SET	Used to hold the properties of an entity.
PROPERTY_SET_COLLECTION	List of <code>PROPERTY_SET</code> structures.
MOD_PROPERTY_SET	Structure to hold modify operations on an entity.

DAS_URL Interface Reference

This chapter describes the Oracle extensions to the DAS_URL Service Interface. It contains these sections:

- [Directory Entries for the Service Units](#)
- [Service Units and Corresponding URL Parameters](#)
- [DAS URL API Parameter Descriptions](#)
- [Search-and-Select Service Units for Users or Groups](#)

Directory Entries for the Service Units

Table 15–1 lists the Oracle Delegated Administration Services units and the directory entries that store relative URLs for these units.

Table 15–1 *Service Units and Corresponding Entries*

Service Unit	Entry
Create User	cn=Create User,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Edit User	cn=Edit User,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Edit User when GUID is passed as a parameter	cn=Edit UserGivenGUID,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Delete User	cn=DeleteUser,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Delete User when GUID of the user to be deleted is passed as a parameter	cn=DeleteUserGivenGUID,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Create Group	cn=Create Group,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Edit Group	cn=Edit Group,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Edit the group whose GUID is passed through a parameter	cn=Edit GroupGivenGUID,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Delete Group	cn=DeleteGroup,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Delete group with the GUID passed through a parameter	cn=DeleteGroupGivenGUID,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext

Table 15–1 (Cont.) Service Units and Corresponding Entries

Service Unit	Entry
Assign privileges to a user	cn=User Privilege,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Assign privileges to a user with the GUID passed through a parameter	cn=User Privilege Given GUID,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Assign privilege to a group	cn=Group Privilege,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Assign privilege to a group with the given GUID	cn=Group Privilege Given GUID,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
View User account information/Profile	cn=Account Info,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Edit User account Information/Profile	cn=Edit My Profile,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Change Password	cn=Password Change,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Search User	cn=User Search,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Search Group	cn=Group Search,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Search User LOV	cn=User LOV,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Search Group LOV	cn=Group LOV,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
EUS Console	cn=EUS Console,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext "
Delegation Console	cn=Delegation Console,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
Password Reset	cn=Reset Password,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext
View User Profile	cn=View User Profile,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext

Service Units and Corresponding URL Parameters

[Table 15–2](#) lists the service units and the URL parameters that can be passed to these units.

Table 15–2 Service Units and Corresponding URL Parameters

Service Unit	Parameter	Return Values
Create User	doneURL homeURL cancelURL enablePA enableHomeURL enableHelpURL	returnGUID

Table 15–2 (Cont.) Service Units and Corresponding URL Parameters

Service Unit	Parameter	Return Values
Edit User	homeURL	-
	doneURL	
	cancelURL	
	enablePA	
	enableHomeURL	
	enableHelpURL	
Edit UserGivenGUID	homeURL	-
	doneURL	
	cancelURL	
	enablePA	
	userGUID	
	enableHomeURL	
Edit My Profile	homeURL	-
	doneURL	
	cancelURL	
	enableHomeURL	
	enableHelpURL	
	Delegation Console	
DeleteUser	homeURL	-
	doneURL	
	cancelURL	
	enableHomeURL	
	enableHelpURL	
	DeleteUserGivenGUID	
doneURL		
cancelURL		
userGUID		
enableHomeURL		
enableHelpURL		
User Privilege	homeURL	-
	doneURL	
	cancelURL	
	enableHomeURL	
	enableHelpURL	
	User Privilege Given GUID	
doneURL		
cancelURL		
userGUID		
enableHomeURL		
enableHelpURL		
Create Group	homeURL	returnGUID
	doneURL	
	cancelURL	
	enablePA	
	parentDN	
	enableHomeURL	
enableHelpURL		

Table 15–2 (Cont.) Service Units and Corresponding URL Parameters

Service Unit	Parameter	Return Values
Edit Group	homeURL	-
	doneURL	
	cancelURL	
	enablePA	
	enableHomeURL	
	enableHelpURL	
Edit Group Given GUID	homeURL	-
	doneURL	
	cancelURL	
	enablePA	
	groupGUID	
	enableHomeURL	
Delete Group	homeURL	-
	doneURL	
	cancelURL	
	enableHomeURL	
	enableHelpURL	
	Delete Group Given GUID	
doneURL		
cancelURL		
groupGUID		
enableHomeURL		
enableHelpURL		
Group Privilege	homeURL	-
	doneURL	
	cancelURL	
	enableHomeURL	
	enableHelpURL	
	Group Privilege Given GUID	
doneURL		
cancelURL		
groupGUID		
enableHomeURL		
enableHelpURL		
Account Info	homeURL	-
	doneURL	
	cancelURL	
	enableHomeURL	
	enableHelpURL	
	Password Change	
doneURL		
cancelURL		
enableHomeURL		
enableHelpURL		
User Search		homeURL
	doneURLm	
	cancelURL	
	enableHomeURL	
	enableHelpURL	

Table 15–2 (Cont.) Service Units and Corresponding URL Parameters

Service Unit	Parameter	Return Values
Group Search	homeURL	-
	doneURL	
	cancelURL	
	enableHomeURL	
	enableHelpURL	
Password Reset	cancelURL	-
	doneURL	
	enableHomeURL	
	enableHelpURL	
View User Profile	userGuid	-
	doneURL	
	homeURL	
	enableHomeURL	
	enableHelpURL	
User LOV	base	userDn
	cfilter	userGuid
	title	userName
	dasdomain	nickName
	callbackURL	userEmail
Group LOV	otype	groupDN
	base	groupGuid
	cfilter	groupName
	title	groupDescription
	dasdomain	
	callbackURL	

DAS URL API Parameter Descriptions

The parameters described in [Table 15–3](#) are used with DAS units.

Table 15–3 DAS URL Parameter Descriptions

Parameter	Description
homeURL	The URL that is linked to the global button Home. When the calling application specifies this value, clicking Home redirects the DAS unit to the URL specified by this parameter.
doneURL	This URL is used by DAS to redirect the DAS page at the end of each operation. In the case of Create User, once the user is created, clicking OK redirects the URL to this location.
callbackURL	DAS uses this URL to send return values to the invoking application. For UserLOV and GroupLOV units, the return values are submitted as HTML form parameters through the HTTP POST method.
cancelURL	This URL is linked with all the Cancel buttons shown in the DAS units. Any time the user clicks Cancel, the page is redirected to the URL specified by this parameter.
enablePA	This parameter takes a Boolean value of true or false. Set to true, the parameter enables the Assign Privileges in User or Group operation. If the enablePA is passed with value of true in the Create User page, the Assign Privileges to User section also appears in the Create User page.
userGUID	This is the GUID of the user to be edited or deleted. This corresponds to the orclguid attribute. Specifying the GUID causes the search for the user step in either editUser or deleteUser units to be skipped.

Table 15–3 (Cont.) DAS URL Parameter Descriptions

Parameter	Description
GroupGUID	This is the GUID of the group to be edited or deleted. This corresponds to the orclguid attribute. Specifying the GUID causes the search for the group step in either editGroup or deleteGroup units to be skipped.
parentDN	When this parameter is specified in CreateGroup, the group is created under this container. If the parameter is not specified, group creation defaults to the group search base.
base	This parameter represents the search base in the case of search operations.
cfilter	This parameter represents the filter to be used for the search. This filter is LDAP compliant.
title	This parameter represents the title to be shown in the Search and Select LOV page.
otype	This parameter represents the object type used for search. Values supported are Select, Edit, and Assign.
returnGUID	This parameter is appended to the done URL in case of a create operation. The value will be the orclguid of the new object.
dasdomain	This parameter is needed only when the browser is Internet Explorer and the calling URL and the DAS URL are on different hosts and in the same domain. An example value is us.oracle.com. Note the calling application also needs to set the document.domain parameter on the formload. For more details, refer to Microsoft support at: http://support.microsoft.com/
enableHomeUR	When this parameter is passed with a value of false, the service unit will be rendered without the home button and home link. By default, the parameter is set to true.
enableHelpURL	When this parameter is passed with a value of false, the service unit will be rendered without the help button and help link. By default, the parameter is set to true.

Search-and-Select Service Units for Users or Groups

DAS provides service units for searching and selecting users or groups. These service units are sometimes referred to as user or group List Of Values (LOV).

Invoking Search-and-Select Service Units for Users or Groups

A custom application can open a popup window and populate its contents by supplying a search-and-select URL for a user or group by using a URL of the form:

```
http://das_host:das_port/oiddas/ui/oracle/ldap/das/search/LOVUserSearch?title=User&callbackurl=http://app_host:app_port/custapp/Callback
```

or

```
http://das_host:das_port/oiddas/ui/oracle/ldap/das/search/LOVGroupSearch?title=User&callbackurl=http://app_host:app_port/custapp/Callback
```

respectively. For example:

```
http://server02.example.com:7777/oiddas/ui/oracle/ldap/das/search/LOVUserSearch?Mary.Smith=User&callbackurl=http://server04.example.com:7778/custapp/Callback
```


In this example, `server02.example.com:7777` is the host name and port of the Oracle Internet Directory DAS application server. `server04.example.com:7778` is the host name and port of the custom application server. `Mary.Smith` is a string that appears in the title of the Search and Select page.

`http://server04.example.com:7778/custapp/Callback` is a URL of the custom application server that receives the selected parameters for users or groups.

Note: To avoid popup blocking, the custom application may open the popup window with a URL on the local custom application server and immediately redirect to the Oracle Internet Directory DAS User or Group Search-and-Select URL.

Receiving Data from the User or Group Search-and-Select Service Units

After a User or Group has been selected via the Oracle Internet Directory DAS User or Group Search-and-Select Service Unit, an HTTP form will be submitted to the `callbackurl` page using the POST method. The parameters defined in [Table 15-4](#) and [Table 15-5](#) are available to the `callbackurl` page:

Table 15-4 *User Search and Select*

Parameter	Description
<code>userDn</code>	User's distinguished name.
<code>userGuid</code>	User's global unique ID.
<code>userName</code>	User's name.
<code>nickName</code>	User's nickname
<code>userEmail</code>	User's email.

Table 15-5 *Group Search and Select*

Parameter	Description
<code>groupDn</code>	Group's distinguished name.
<code>groupGuid</code>	Group's global unique ID.
<code>groupName</code>	Group's name.
<code>groupDescription</code>	Group's description.

The `callbackurl` page in the popup window may transfer the form parameters to the invoking page in the opener window using JavaScript. It may then close the popup window.

Note: To avoid JavaScript security problems, the custom application may supply the `callbackurl` page on the same server as the invoking page. This enables the `callbackurl` page in the popup window and the invoking page in the opener window to communicate directly through JavaScript.

Centralized User Provisioning Java API Reference

As of 10g Release 2 (10.1.2.0.2), the Oracle Internet Directory SDK includes a centralized user provisioning API, which enables you to manage users and their application properties in the Oracle Identity Management infrastructure. This chapter describes the main features of the API and explains how to use them.

Reorder App first, then user.

This chapter contains the following sections:

- [Application Configuration](#)
- [User Management](#)
- [Debugging](#)
- [Sample Code](#)

Application Configuration

Applications must register with the provisioning system in order to be recognized as provisionable. They must also create their own configuration in Oracle Internet Directory using the command-line interface. Java classes exist for viewing application configurations.

This section contains the following topics:

- [Application Registration and Provisioning Configuration](#)
- [Application Configuration Classes](#)

Application Registration and Provisioning Configuration

In order to register with the provisioning system, an application must create a provisioning configuration. Once the provisioning configuration exists, the provisioning system identifies the application as directory-enabled and provisionable.

The application must perform the following steps to create a provisioning configuration:

1. [Application Registration](#)
2. [Provisioning Configuration](#)

Application Registration

Oracle applications typically register themselves by using the repository APIs in the `repository.jar` file under `$ORACLE_HOME/jlib`. This file is provided during installation specifically for application registration. In addition to creating an application entry in Oracle Internet Directory, repository APIs can be used to add the application to privileged groups.

Applications written by customers, however, cannot use the `repository.jar` APIs to perform application registration. So application developers must use LDIF templates and create application entries in Oracle Internet Directory using LDAP commands.

An application must create a container for itself under one of these containers:

- `"cn=Products,cn=OracleContext"`—for applications that service users in multiple realms
- `"cn=Products,cn=OracleContext,RealmDN"`—for applications that service users in a specific realm

If an application is configured for a specific realm, then that application cannot manage users in other realms. In most cases, you should create the application outside any identity management realm so that the application is not tied to a specific realm in Oracle Internet Directory.

Whenever a new instance of the application installs, a separate entry for the application instance is created under the application's container. Some of the provisioning configuration is common to all the instances of a particular type and some is specific to the instance. When multiple instances of an application are deployed in an enterprise, each instance is independent of the others. Each instance is defined as a separate provisionable application. Users can be provisioned for one or more instances of this application, so that the user can get access to one or more instances of this application.

The examples in this section are for a sample application similar to Oracle Files. When the first instance of this application installs, specific entries must be created in Oracle Internet Directory. In the following example, the name of this application, chosen at run time, is `Files-App1` and the type of the application is `FILES`. The application can have LDIF templates that can be instantiated if required and then uploaded to Oracle Internet Directory. In this example, the application identity is outside any realm. That is, it is under the `"cn=Products,cn=OracleContext"` container.

```
dn: cn=FILES,cn=Products,cn=OracleContext
changetype: add
objectclass: orclContainer
```

```
dn: orclApplicationCommonName=Files-App1,cn=FILES,cn=Products,cn=OracleContext
changetype: add
orclappfullname: Files Application Instance 1
userpassword: welcome123
description: This is a test Application instance.
protocolInformation: xxxxx
orclVersion: 1.0
orclaci: access to entry by group="cn=odisgroup,cn=DIPAdmins,
cn=Directory Integration Platform,cn=Products,
cn=OracleContext" (browse,proxy) by group="cn=User Provisioning Admins,
cn=Groups,cn=OracleContext" (browse,proxy)
orclaci: access to attr=(*) by group="cn=odisgroup,cn=DIPAdmins,
cn=Directory Integration Platform,cn=Products,
cn=OracleContext" (search,read,write,compare)
```

```
by group="cn=User Provisioning Admins,
cn=Groups,cn=OracleContext" (search,read,write,compare)
```

The ACLs shown in the example are discussed in the "[Application User Data Location](#)" section.

The application is expected to grant certain privileges to some provisioning services as well as provisioning administrators.

When the second instance of this application installs, the following entries must be created in Oracle Directory Integration and Provisioning, assuming the name of this application, decided at run time, is `Files-App2`.

```
dn: orclApplicationCommonName=Files-App2,cn=FILES,cn=Products,cn=OracleContext
changetype: add
orclappfullname: Files Application Instance 2
userpassword: welcome123
description: This is a test Application instance.
orclVersion: 1.0
orclaci: access to entry by group="cn=odisgroup,
cn=DIPAdmins,cn=Directory Integration Platform,cn=Products,
cn=OracleContext" (browse,proxy) by group="cn=User Provisioning Admins,
cn=Groups,cn=OracleContext" (browse,proxy)
orclaci: access to attr=(*) by group="cn=odisgroup,cn=DIPAdmins,
cn=Directory Integration Platform,cn=Products,
cn=OracleContext" (search,read,write,compare) by
group="cn=User Provisioning Admins,cn=Groups,cn=OracleContext"
(search,read,write,compare)
```

Once the application creates its entries successfully, the application's identity is registered in Oracle Internet Directory. At this point, the application can add itself to certain privileged groups in Oracle Internet Directory, if it needs specific privileges. [Table 16-1, "Some Useful Privilege Groups"](#) shows some of the privileged groups that an application can add itself to. Each of these groups exists in every realm and also in the `RootOracleContext`. The `RootOracleContext` Group is a member of the group in all the realms

Table 16-1 Some Useful Privilege Groups

Group Name	Privilege
OracleDASCreateUser	Create a public user
OracleDASEditUser	Edit a public user
OracleDASDeleteUser	Delete a public user
OracleDASCreateGroup	Create a new public group
OracleDASEditGroup	Edit a public group
OracleDASDeleteGroup	Delete a public group

For example, the following LDIF file adds the `Files-App1` application to `cn=OracleCreateUser`, which gives it the privilege to create users in all realms.

```
dn: cn=OracleCreateUser,cn=Groups,cn=OracleContext
changetype: modify
add: uniquemember
uniquemember:
orclApplicationCommonName=Files-App1,cn=FILES,cn=Products,cn=OracleContext
```

Provisioning Configuration

An application's provisioning configuration is maintained in its provisioning profile. The provisioning system supports three different provisioning profile versions: Versions 1.1, 2.0 and 3.0. The provisioning service provides different service for the different profile version. Some generic configuration details are common to all applications, regardless of version.

Differences Between Provisioning Configuration Versions

The differences between the Version 3.0 profile and the Version 2.0 and Version 1.1 profiles are as follows:

- The new provisioning framework recognizes only Version 3.0 applications. Therefore, only applications with provisioning profile Version 3.0 show up as target applications to be provisioned in Oracle Provisioning Console. Applications with Version 2.0 and Version 1.1 profiles do not show them up as applications to be provisioned in the Provisioning Console. Still, the applications are notified about the events that the applications have configured for.
- Creating the provisioning configuration of an application is a multi step process for Version 3.0 profiles. For the earlier version profiles, provisioning registration requires only a single step, running the `oidprovtool` command.
- Applications can subscribe for provisioning events using different interfaces. Two of the interfaces, Java and OID-LDAP, are available only for interface Version 3.0, which is coupled with provisioning configuration Version 3.0. See [Table 16-2, "Interfaces and Their Configuration"](#).
- An application can specify its application-specific user attributes configuration in an LDIF file. This is supported only for interface Version 3.0, which is coupled with provisioning configuration Version 3.0. See ["Application User Attribute and Defaults Configuration"](#) on page 16-9
- The provisioning status of the user, discussed in the *Oracle Identity Management Integration Guide*, is maintained only for Version 3.0 applications. It is not maintained for applications having profiles earlier than Version 3.0.
- Event propagation configuration parameters vary from one version to another. See [Table 16-5, "Event propagation parameters"](#).

Version 3.0-Specific Provisioning Configuration

Unless otherwise stated, the remainder of this section describes the Version 3.0-specific provisioning configuration. [Figure 16-1](#) shows the DIT in Oracle Internet Directory used to store the provisioning configuration. All the provisioning configuration information is located under the following container:

```
cn=Provisioning,cn=Directory Integration Platform,cn=Products,cn=OracleContext
```

Common provisioning configuration information is stored in entries under the container:

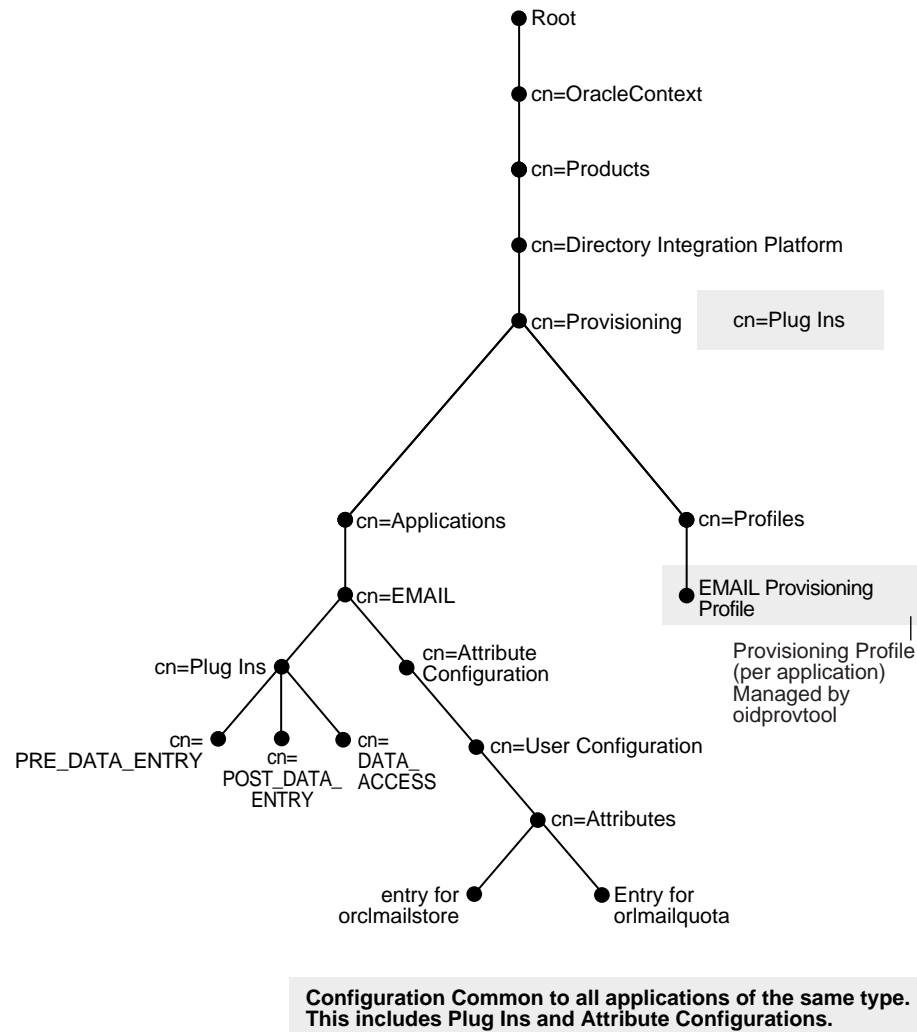
```
cn=Profiles,cn=Provisioning,cn=Directory Integration Platform,
cn=Products,cn=OracleContext
```

The rest of the provisioning configuration for an application is located under:

```
cn=ApplicationType,cn=Applications,cn=Provisioning,
cn=Directory Integration Platform,cn=Products,cn=OracleContext
```

All the instances of a specific application type share the configuration under this container. That is, whenever a second instance of an existing application type creates a provisioning profile, all the configuration information under the "cn=ApplicationType" container is shared.

Figure 16–1 The Directory Information Tree for Provisioning Configuration Data



The Profiles container contains the following types of configuration information:

- [Application Identity Information](#)
- [Application Identity Realm Information](#)
- [Application Provisioning and Default Policy](#)
- [Application User Data Location](#)
- [Event Interface Configuration](#)
- [Application User Attribute and Defaults Configuration](#)
- [Application Provisioning Plug-in Configuration](#)
- [Application Propagation Configuration](#)
- [Application Event Propagation Run Time Status](#)

Whenever an instance of an application creates a profile, the new profile is stored as a separate entry under the `Profiles` container in the following naming format:

```
orclODIPProfileName=GUID_of_the_Realm_Entry_GUID_of_the_Application_Identity,...
```

An application must specify the following information when creating a provisioning configuration:

Application Identity Information An instance of an application is uniquely identified by the following parameters:

- **Application DN**—A unique DN in the Oracle Internet Directory representing the application. This is a mandatory parameter.
- **Application Type**— A parameter that is common to all instances of the same application. Multiple instances of a particular type can share some configuration. This is a mandatory parameter.
- **Application Name**—This can be separately specified. If not specified, it is extracted from the DN. This is an optional parameter.
- **Application Display Name**—A user-friendly name for the application. This shows up on the Provisioning Console as a target provisionable application. This is an optional parameter.

You provide these application identity parameters while creating the provisioning profile by using the following arguments to the `$ORACLE_HOME/bin/oidprovtool` command line utility, respectively:

- `application_type`
- `application_dn`
- `application_name`
- `application_display_name`

See Also: The `oidprovtool` command-line tool reference in *Oracle Identity Management User Reference*

Application Identity Realm Information An application registers for a specific realm in order to provide services to the users of that realm only. An application must create a separate provisioning profile for each of the realms it provides services for. In a multi realm scenario, such as a hosted OracleAS Portal scenario, applications must register for individual realms.

Whenever a provisioning administrator for a realm accesses the Provisioning Console, only the applications that are registered for that realm are shown as provisionable target applications.

The application specifies realm information while creating the provisioning profile by using the `$ORACLE_HOME/bin/oidprovtool` command line utility with the argument `organization_dn`.

See Also: The `oidprovtool` command-line tool reference in *Oracle Identity Management User Reference*.

Application Provisioning and Default Policy While creating a provisioning profile, an application can specify whether the Provisioning Console should manage provisioning to that application or not. If not, the application does not show up on the Provisioning

Console as an application to be provisioned. However, Oracle Directory Integration and Provisioning still processes this profile and propagates the events as expected.

An application specifies this information while creating the provisioning profile by using the `application_isdasvisible` argument to the `$ORACLE_HOME/bin/oidprovtool` command line utility. The default value is `TRUE`.

An application can configure a default policy determining whether all the users in that realm should be provisioned for that application by default or no users should be provisioned by default. The valid values are

- `PROVISIONING_REQUIRED`—all users will be provisioned by default
- `PROVISIONING_NOT_REQUIRED`—no users will be provisioned by default

The default is set to `PROVISIONING_REQUIRED`

You can override the default policy with application-provided policy plug-ins at run time. In addition, an administrator can override both the default policy and the decision of the policy plug-in.

An application provides the default policy information by using the `default_provisioning_policy` argument to the `$ORACLE_HOME/bin/oidprovtool` command line utility.

Application User Data Location Application-specific user information is stored in the application-specific containers. If this data is to be managed by the provisioning system, the application must specify the location of these containers during provisioning registration. An application specifies its user data location by using the `user_data_location` argument to the `$ORACLE_HOME/bin/oidprovtool` command line utility. The application must ensure that the ACLs on this container allow Oracle Delegated Administration Services and Oracle Directory Integration and Provisioning to manage the information in this container.

Event Interface Configuration Applications can subscribe for provisioning events using different interfaces: PLSQL, Java, and OID-LDAP. [Table 16-2, "Interfaces and Their Configuration"](#) lists the supported interfaces and their associated configuration. Note that `INTERFACE_VERSION` is coupled with provisioning profile version.

Table 16-2 Interfaces and Their Configuration

Configuration Parameter	PLSQL	Java	OID-LDAP
<code>INTERFACE_VERSION</code>	1.1, 2.0, 3.0	3.0	3.0
<code>INTERFACE_NAME</code>	The name of the PLSQL package that implements the Interface	Not used	Not used
<code>INTERFACE_CONNECT_INFO</code>	The Database Connect String. Multiple formats supported for all versions.	Not used	Not used
<code>INTERFACE_ADDITIONAL_INFO</code>	Not used	Not used	Not used
Plugin types	<code>PRE_DATA_ENTRY</code> , <code>POST_DATA_ENTRY</code> , <code>DATA_ACCESS</code>	<code>PRE_DATA_ENTRY</code> , <code>POST_DATA_ENTRY</code> , <code>DATA_ACCESS</code> , <code>EVENT_DELIVERY</code> (MUST)	<code>PRE_DATA_ENTRY</code> , <code>POST_DATA_ENTRY</code> , <code>DATA_ACCESS</code>

Table 16–2 (Cont.) Interfaces and Their Configuration

Configuration Parameter	PLSQL	Java	OID-LDAP
Description	Mainly for applications that have an Oracle Database backend. The DIP Server pushes the event to the remote Database by invoking the PLSQL procedure.	If the Interface Type is JAVA, an event delivering plug-in must be configured or the server will give errors. The plug-in configuration determines the rest of the configuration. See Application Provisioning Plug-in Configuration .	Mainly used in cases where the application is very tightly bound to Oracle Internet Directory and event delivery through the PLSQL interface or the JAVA Event Delivery Plug-in is unnecessary. This interface will be deprecated in future. Please use the JAVA Interface instead.

Applications can use the following arguments to \$ORACLE_HOME/bin/oidprovtool when specifying an event interface configuration:

- interface_type (Default is PLSQL)
- interface_version (Default is 2.0)
- interface_name
- interface_connect_info
- interface_additional_info

[Table 16–3, "Information Formats Supported by the PLSQL Interface"](#) lists the interface connection information formats that the PL/SQL interface supports when it connects to a remote database. All the formats are supported for all interface versions.

Table 16–3 Information Formats Supported by the PLSQL Interface

Format	Description
<i>dbHost:dbPort:dbSID:username:password</i>	Old format, not recommended. Oracle Directory Integration and Provisioning passes this to the thin JDBC Driver.
<i>dbHost:dbPort:dbServiceName:username:password</i>	Newer format. Not Recommended for High Availability implementations, as the database host and port might change in such scenarios. DIP passes this to the thin JDBC Driver.
<i>DBSVC=DB_TNS_Connect_Sring_Alias:username:password</i>	Used by JDBC thick OCI Driver. The local <code>tnsnames.ora</code> file must contain this alias on the node where DIP is running.
<i>DBURL=ldap://LDAP_host:LDAP_port/ServiceName,cn=OracleContext</i>	Recommended format, as it takes care of High Availability requirements. DIP passes this to the thin JDBC Driver and the driver looks up the Database Registration entry in Oracle Internet Directory to get the actual Database connection information.

Some examples of supported formats are:

```
localhost:1521:iasdb:scott:tiger
```

```
localhost:1521:iasdbsvc:scott:tiger
```

```
DBSVC=TNSSALIAS:scott:tiger
```

```
DBURL=ldap://acme.com:389/sampleldbname:scott:tiger
```

Application User Attribute and Defaults Configuration An application can specify its application-specific user attributes configuration in an LDIF file. This is supported only for interface version 3.0.

As shown in [Figure 16–1, "The Directory Information Tree for Provisioning Configuration Data"](#), the configuration for a particular attribute is stored as a separate entry under the container:

```
"cn=Attributes,cn=User Configuration,cn=Attribute configuration,
cn=Application_Type,cn=Applications,cn=Provisioning,
cn=Directory Integration Platform,cn=Products,cn=OracleContext"
```

There is no argument to `oidprovtool` for uploading this information. The application must use an LDAP file and command-line tools to upload its attribute configuration information to Oracle Internet Directory.

Each application-specific attribute is represented as a separate entry. The following example is for the attribute `orclFilesDomain`:

```
dn: cn=orclFilesDomain,cn=Attributes,cn=User configuration,cn=Attribute
configuration,.....
changetype: add
orclDasAdminModifiable: 1
orclDasViewable: 1
displayname: Files Domain
orclDasMandatory: 1
orclDasUIType: LOV
orclDasLOV: us.oracle.com
orclDasLOV: oraclecorp.com
orclDASAttrIsUIField: 1
orclDASAttrIsFieldForCreate: 1
orclDASAttrIsFieldForEdit: 1
orclDASAttrToDisplayByDefault: 1
orclDASSelfModifiable: 1
orclDASAttrDisplayOrder: 1
orclDASAttrDefaultValue: oraclecorp.com
orclDASAttrObjectClass: orclFILESUser
objectclass: orclDASConfigAttr
```

[Table 16–4, " Properties Stored as Attributes in the Attribute Configuration Entry"](#) explains the significance of each of the properties that are stored as attributes in the attribute configuration entry.

Table 16–4 Properties Stored as Attributes in the Attribute Configuration Entry

Property Name	Description	Comments
<code>orclDASIsUIField</code>	Whether this property is to be shown in the DAS Console or not	Not Used in 10g Release 2 (10.1.2). All attributes are shown.
<code>orclDASUIType</code>	The Type of the UI Field: singletext, multitext, LOV, DATE, Number, password	Used by Oracle Internet Directory Self-Service Console only
<code>orclDASAdminModifiable</code>	Whether the field is modifiable by the administrator or not	Not Used in 10g Release 2 (10.1.2). All attributes are modifiable by administrator.

Table 16–4 (Cont.) Properties Stored as Attributes in the Attribute Configuration Entry

Property Name	Description	Comments
orclDASViewAble	Whether this attribute is a read-only attribute in the Oracle Internet Directory Self-Service Console	Not Used in 10g Release 2 (10.1.2)
displayName	The Localized Name of the attribute as it shows on the Oracle Internet Directory Self-Service Console	
orclDASIsMandatory	Whether this attribute is mandatory or not	If a mandatory attribute is not populated, the Oracle Internet Directory Self-Service Console complains
orclDASAttrIsFieldForCreate	Whether to expose this attribute only during user creation	Not Used in 10g Release 2 (10.1.2)
orclDASAttrIsFieldForEdit	Whether to expose this attribute only during user editing	Not Used in 10g Release 2 (10.1.2)
orclDASAttrToDisplayByDefault	Whether to hide the attribute by default under a collapsed section	Not Used in 10g Release 2 (10.1.2)
orclDASSelfModifiable	Whether this attribute is modifiable by the user or not	Not Used in 10g Release 2 (10.1.2), as Oracle Internet Directory Self-Service Console is only for application-specific attributes. Users cannot change their user preferences from the Oracle Internet Directory Self-Service Console.
OrclDASAttrDisplayOrder	The order in which the attribute is to be displayed in the application-specific section	Not Used in 10g Release 2 (10.1.2)
OrclDASAttrDefaultValue	The initial default value for the attribute that is used by the provisioning components: Oracle Internet Directory Self-Service Console, Oracle Directory Integration and Provisioning, Bulk Provisioning Tool	Can be changed using the Oracle Internet Directory Self-Service Console <i>Application Management</i> Page. The Plug-ins or the administrator can override the initial default values.
OrclDASAttrObjectClass	The LDAP object class that the attribute belongs to.	Used to create the application-specific user entries that the provisioning system maintains.

If an application has application-specific attributes, you can specify that the provisioning system manage its attributes defaults. You do that by using the `manage_application_defaults` argument to `$ORACLE_HOME/bin/oidprovtool`. This argument is `TRUE` by default.

Application Provisioning Plug-in Configuration Application provisioning plug-ins are discussed in

[Appendix A, "Java Plug-ins for User Provisioning"](#).

Application Propagation Configuration Event propagation configuration parameters vary from one profile version to another. [Table 16–5, "Event propagation parameters"](#) lists and describes configuration parameters for event propagation.

Table 16–5 Event propagation parameters

Parameter	Supported Provisioning Profile Version	Description
profile_mode	2.0,3.0	Whether the application is to receive outbound provisioning events from Oracle Internet Directory, to send inbound events, or both. Values are OUTBOUND (default), INBOUND, and BOTH.
Schedule	1.1, 2.0, 3.0	The scheduling interval after which pending events are propagated
enable_bootstrap	3.0	Enables events for application bootstrapping. This specifies that the application should be notified of users that existed in Oracle Internet Directory before the application created its provisioning profile.
enable_upgrade	3.0	Enables events for application user upgrade. This specifies that the application should be notified of users that existed in Oracle Internet Directory before the upgrade. If the application was present before the upgrade, users might already exist in the application. For such users, Oracle Directory Integration and Provisioning sends an Upgrade Event to the application so that the user is handled differently from a normal new user.
lastchangenumber	3.0	The change number in Oracle Internet Directory from which the events need to be sent to the application.
max_prov_failure_limit	3.0	The maximum number of retries that the Oracle Directory Integration and Provisioning server attempts when provisioning a user for that application.
max_events_per_invocation	2.0, 3.0	For bulk event propagation, this specifies the maximum number of events that can be packaged and sent during one invocation of the event interface.
max_events_per_schedule	2.0	Maximum number of events that Oracle Directory Integration and Provisioning sends to an application in one execution of the profile. The default is 25. In deployments with many profiles and applications, this enables Oracle Directory Integration and Provisioning, which is multithreaded, to execute threads for multiple profiles.
event_subscription	1.1, 2.0, 3.0	<p>Defines the types of OUTBOUND events an application is to receive from the event propagation service. The format is:</p> <p><i>Object_Type:Domain:Operation(Attributes,...)</i></p> <p>For example:</p> <p><code>USER:cn=users,dc=acme,dc=com:ADD(*)</code></p> <p>specifies that USER_ADD event should be sent if the user that was created is under the specified domain and that all attributes should also be sent.</p> <p><code>USER:cn=users,dc=acme,dc=com:MODIFY(cn,sn.mail,telephonenumber)</code></p> <p>specifies that USER_MODIFY event should be sent if the user that was modified is under the specified domain and any of the listed attributes were modified</p> <p><code>USER:cn=users,dc=acme,dc=com:DELETE</code></p> <p>specifies that USER_DELETE event should be sent if a user under the specified domain was deleted</p>

Table 16–5 (Cont.) Event propagation parameters

Parameter	Supported Provisioning Profile Version	Description
event_permitted_operations	2.0	<p>Defines the types of INBOUND events an application is privileged to send to the Oracle Directory Integration and Provisioning server. The format is:</p> <p><i>Object_Type:Domain:Operation(Attributes,...)</i></p> <p>For example:</p> <p><code>IDENTITY:cn=users,dc=acme,dc=com:ADD(*)</code></p> <p>specifies that <code>IDENTITY_ADD</code> event is allowed for the specified domain and all attributes are also allowed. This means that the application is allowed to create users in Oracle Internet Directory.</p> <p><code>IDENTITY:cn=users,dc=acme,dc=com:MODIFY(cn,sn.mail,telephonenumber)</code></p> <p>Specifies that <code>IDENTITY_MODIFY</code> is allowed for only the attributes in the list. Other attributes are silently ignored. This means that the application is allowed to modify the listed attributes of the users in Oracle Internet Directory.</p> <p><code>IDENTITY:cn=users,dc=acme,dc=com:DELETE</code></p> <p>Specifies that the application is allowed to delete users in Oracle Internet Directory</p>
event_mapping_rules	2.0	<p>For INBOUND profiles, this specifies the type of object received from an application and a qualifying filter condition to determine the domain of interest for this event. Multiple rules are allowed. The format is:</p> <p><i>Object_Type: Filter_condition: Domain_Of_Interest</i></p> <p>For example:</p> <p><code>EMP::cn=users,dc=acme,dc=com</code></p> <p>specifies that if the object type received is <code>EMP</code>, the event is meant for the domain "<code>cn=users,dc=acme,dc=com</code>".</p> <p><code>EMP:l=AMERICA:l=AMER,cn=users,dc=acme,dc=com</code></p> <p>specifies that if the object type received is <code>EMP</code>, and the event has the attribute <code>l</code> (locality) and its value is <code>AMERICA</code>, the event is meant for the domain "<code>l=AMER,cn=users,dc=acme,dc=com</code>".</p>

Application Event Propagation Run Time Status The Oracle Provisioning Service records a user's provisioning status in Oracle Internet Directory for each provisioning-integrated application. This is described in the Deploying and Configuring Provisioning chapter of *Oracle Identity Management Integration Guide*.

Application Configuration Classes

The `oracle.idm.user.provisioning.configuration.Configuration` class enables you to obtain provisioning schema information. The `oracle.idm.user.provisioning.configuration.Application` class enables you to obtain metadata for registered applications. These classes are documented under the package `oracle.idm.provisioning.configuration`.

The `Configuration` class provides access to application configurations. To construct, a `Configuration` object, you must specify the realm. For example:

```
Configuration cfg = new Configuration ("us");
```

Then you use `Configuration` class methods to get one or all application configurations in a realm. You must supply the LDAP context of the realm.

The `Configuration` object is a fairly heavy weight object, as its creation requires access to the Oracle Internet Directory metadata. Best practice is to create a `Configuration` object once during initialization of an application, then to reuse it for all operations that require it.

The `Application` object represents an application instance. Its methods provide metadata about a registered application in the infrastructure.

User Management

When Oracle Directory Integration and Provisioning or Oracle Delegated Administration Services invokes a provisioning plugin, it passes information about the user being provisioned. A deployed application can use the user object to modify the user.

The user management provisioning classes provide the following operations:

- Create, modify, and delete a base user
- Create, modify, and delete application-specific user information
- Search base users
- Retrieve user provisioning status for applications

This section includes the following topics:

- [Creating a User](#)
- [Modifying a User](#)
- [Deleting a User](#)
- [Looking Up a User](#)

Creating a User

Creating a user in the Oracle Identity Management repository consists of two steps:

1. Creating basic user information in the specified realm. This information is referred to as the base user.
2. Creating the application-specific user attributes, or footprint. This information is referred to as the application user.

The combination of the base user and application user in the repository is referred to as the Oracle Identity Management user. Some methods create only the base user and other create both components of the Oracle Identity Management user.

The minimum information required to create a user is a set of attributes representing the base user. The attributes are in the form of name-value pairs. These user attributes are represented as Java objects using the class `oracle.ldap.util.ModPropertySet`.

Some user creation methods require you to specify the DN of the entry that you want to create in the Oracle Identity Management user repository. Other methods do not require the DN. Instead, they construct the Oracle Identity Management user using the metadata configuration information from the Realm in which the user is created.

If the creation of the base user and application user succeeds, then the creation method returns an `IdmUser` object. You use this object to manage the attributes of the base user and application user.

Modifying a User

Modifying a base user in the Oracle Identity Management repository results in

- Modifying the base user information
- Creating or modifying application user information

You must supply the following information in order to modify an Oracle Identity Management user:

1. The user's DN, GUID, or `IdmUser` object reference
2. The desired changes to the base user attributes, represented as an `oracle.ldap.util.ModPropertySet`

Some user modification methods modify only the base user attributes. Others modify the application user attributes as well.

Deleting a User

Deleting a base user in the Oracle Identity Management repository produces the following results:

- Deleting the base user information
- Deleting the application user information

To modify an Oracle Identity Management user, you must supply the DN, GUID, or `IdmUser` object reference.

As result of this operation, the base user and the application user attributes are deleted.

Looking Up a User

The lookup methods provide two lookup options:

- Look up a specific Oracle Identity Management user using GUID or DN
- Look up a set of Oracle Identity Management users using a search filter

In order to look up Oracle Identity Management users, you must provide the DN or GUID.

The output of a lookup method is one of the following:

- A single `IdmUser` object
- A list of `IdmUser` objects

Debugging

Set `UtilDebug.MODE_PROVISIONING_API` mode to enable debugging and trace information. If you do not specify an output stream for the log messages, they are written to standard output.

The following snippet shows how to set `UtilDebug.MODE_PROVISIONING_API` mode and specify an output stream:


```

import oracle.ldap.util.UtilDebug;
FileOutputStream logStream = new FileOutputStream("ProvAPI.log")
...
UtilDebug.setDebugMode(UtilDebug.MODE_PROVISIONING_API);
UtilDebug.setPrintStream(logStream);

```

Sample Code

The following code example shows how to create, modify, and look up a user and how to get user provisioning status for an application.

```

UtilDebug.setDebugMode(UtilDebug.MODE_PROVISIONING_API);
...
Configuration cfg = new Configuration(realm);
try {
    debug("Connecting...");
    InitialLdapContext ctx =
        ConnectionUtil.getDefaultDirCtx(hostName, port, bindDn, passwd);
    debug("Connected...");
    UserFactory factory = UserFactoryBuilder.createUserFactory(ctx, cfg);

    // Create
    ModPropertySet mpSet = new ModPropertySet();
    mpSet.addProperty("cn", "Heman");
    mpSet.addProperty("sn", "The Master");
    mpSet.addProperty("uid", "Heman");
    IdmUser idmUser = factory.createUser(mpSet);

    // Modify
    mpSet = new ModPropertySet();
    mpSet.addProperty(LDIF.ATTRIBUTE_CHANGE_TYPE_REPLACE, "sn",
        "Heman The Master");
    mpSet.addProperty("givenName", "Master of the Universe");
    factory.modifyUser(idmUser, mpSet);

    // Lookup
    List users = factory.searchUsers(Util.IDTYPE_SIMPLE, "Hema*", null);
    ...

    // Get user provisioning status for an application.
    Application app = cfg.getApplication(lCtx, "Files", "FilesInstance");
    String status = idmUser.getProvisioningStatus(app);

    // Another way to get user provisioning status
    String userDn = idmUser.getDn();
    String status = ProvUtil.getUserProvisioningStatus(dirctx,
        Util.IDTYPE_DN, userDn, app.getType(), app.getName());
} catch (Exception ex) {
    ex.printStackTrace();
    //
}

```

Provisioning Integration PL/SQL API Reference

This chapter examines the registration API for the Oracle Directory Provisioning Integration Service. It contains the following sections:

- [Versioning of Provisioning Files and Interfaces](#)
- [Extensible Event Definition Configuration](#)
- [Inbound and Outbound Events](#)
- [PL/SQL Bidirectional Interface \(Version 3.0\)](#)
- [PL/SQL Bidirectional Interface \(Version 2.0\)](#)
- [Provisioning Event Interface \(Version 1.1\)](#)

Versioning of Provisioning Files and Interfaces

In release 9.0.2, the default interface version was version 1.1. In releases 9.0.4 and 10.1.2.0.0, the interface version defaults to version 2.0. Release 10.1.2.0.1 adds yet a third version. The administrator can use any one of these.

Extensible Event Definition Configuration

This feature is only for outbound events. It addresses the ability to define a new event at run time so that the provisioning integration service can interpret a change in Oracle Internet Directory and determine whether an appropriate event is to be generated and propagated to an application. The following events will be the only configured events at installation time.

An event definition (entry) consists of the following attributes.

- **Event object type** (`orclODIPProvEventType`): This specifies the type of object the event is associated with. For example, the object could be a `USER`, `GROUP`, or `IDENTITY`.
- **LDAP change type** (`orclODIPProvEventChangeType`): This indicates that all kinds of LDAP operations can generate an event for this type of object. (e.g `ADD`, `MODIFY`, `DELETE`)
- **Event criteria** (`orclODIPProvEventCriteria`): The additional selection criteria that qualify an LDAP entry to be of a specific object type. For example, `Objectclass=orclUserV2` means that any LDAP entry that satisfies this criteria can be qualified as this Object Type and any change to this entry can generate appropriate events.

The object class that holds these attributes is `orclODIPProvEventTypeConfig`. The container `cn=ProvisioningEventTypeConfig,cn=odi,cn=oracle internet directory` is used to store all the event type configurations.

Table 17-1 lists the event definitions predefined as a part of the installation.

Table 17-1 Predefined Event Definitions

Event Object Type	LDAP Change Type	Event Criteria
ENTRY	ADD	objectclass=*
	MODIFY	
	DELETE	
USER	ADD	objectclass=interorgperson objectclass=orcluserv2
	MODIFY	
	DELETE	
IDENTITY	ADD	objectclass=interorgperson objectclass=orcluserv2
	MODIFY	
	DELETE	
GROUP	ADD	objectclass=orclgroup objectclass=groupofuniquenames
	MODIFY	
	DELETE	
SUBSCRIPTION	ADD	objectclass=orclservicereceipient
	MODIFY	
	DELETE	
SUBSCRIBER	ADD	objectclass=orclsubscriber
	MODIFY	
	DELETE	

The container `cn=ProvisioningEventTypeConfig,cn=odi,cn=oracle internet directory` is used to store all the event definition configurations. LDAP configuration of the predefined event definitions is as follows:

```
dn: orclODIPProvEventObjectType=ENTRY,cn=ProvisioningEventTypeConfig,cn=odi,
cn=oracle internet directory
orclODIPProvEventObjectType: ENTRY
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=*
objectclass: orclODIPProvEventTypeConfig
```

```
dn:
orclODIPProvEventObjectType=USER,cn=ProvisioningEventTypeConfig,cn=odi,cn=oracle
internet directory
orclODIPProvEventObjectType: USER
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=InetOrgPerson
orclODIPProvEventCriteria: objectclass=orcluserv2
objectclass: orclODIPProvEventTypeConfig
```

```
dn: orclODIPProvEventObjectType=IDENTITY,cn=ProvisioningEventTypeConfig,cn=odi,
cn=oracle internet directory
orclODIPProvEventObjectType: IDENTITY
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
```

```

orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=inetorgperson
orclODIPProvEventCriteria: objectclass=orcluser2
objectclass: orclODIPProvEventTypeConfig

```

```

dn: orclODIPProvEventObjectType=GROUP,cn=ProvisioningEventTypeConfig,cn=odi,
cn=oracle internet directory
orclODIPProvEventObjectType: GROUP
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=orclgroup
orclODIPProvEventCriteria: objectclass=groupofuniquenames
objectclass: orclODIPProvEventTypeConfig

```

```

dn:
orclODIPProvEventObjectType=SUBSCRIPTION,cn=ProvisioningEventTypeConfig,cn=odi,
cn=oracle internet directory
orclODIPProvEventObjectType: SUBSCRIPTION
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=orclservicerecipient
objectclass: orclODIPProvEventTypeConfig

```

```

dn: orclODIPProvEventObjectType=SUBSCRIBER,cn=ProvisioningEventTypeConfig,cn=odi,
cn=oracle internet directory
orclODIPProvEventObjectType: SUBSCRIBER
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=orclsubscriber
objectclass: orclODIPProvEventTypeConfig

```

To define a new event of Object type XYZ (which is qualified with the object class objXYZ), create the following entry in Oracle Internet Directory. The DIP server recognizes this new event definition and propagates events if necessary to applications that subscribe to this event.

```

dn: orclODIPProvEventObjectType=XYZ,cn=ProvisioningEventTypeConfig,cn=odi,
cn=oracle internet directory
orclODIPProvEventObjectType: XYZ
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=objXYZ
objectclass: orclODIPProvEventTypeConfig

```

This means that if an LDAP entry with the object class objXYZ is added, modified, or deleted, DIP will propagate the XYZ_ADD, XYZ_MODIFY, or XYZ_DELETE event to any application concerned.

Inbound and Outbound Events

An application can register as a supplier as well as a consumer of events. The provisioning subscription profile has the attributes described in [Table 17-2](#) on page 17-4.

Table 17–2 Attributes of the Provisioning Subscription Profile

Attribute	Description
EventSubscriptions	<p>Outbound events only (multivalued).</p> <p>Events for which DIP should send notification to this application. The format of this string is [USER]GROUP]:[domain_of_interest]:[DELETE ADD MODIFY(list_of_attributes_separated_by_comma)]</p> <p>Multiple values may be specified by listing the string multiple times, each time with different values. If parameters are not specified, the following defaults are assumed: USER:organization_DN:DELETEDGROUP:organization_DN:DELETE—that is, send user and group delete notifications under the organization DN.</p>
MappingRules	<p>Inbound events Only (multivalued).</p> <p>This attribute is used to map the type of object received from an application and a qualifying filter condition to determine the domain of interest for this event. The mapping takes this form:</p> <p><i>OBJECT_TYPE: Filter_condition: domain_of_interest</i></p> <p>Multiple rules are allowed. In the mapping EMP:cn=users,dc=acme,dc=com, the object type received is EMP. The event is meant for the domain cn=users,dc=acme,dc=com. In the mapping EMP:l=AMERICA:l=AMER,cn=users,dc=acme,dc=com, the object type received is EMP. The event is meant for the domain l=AMER,cn=users,dc=acme,dc=com.</p>
permittedOperations	<p>Inbound events only (multi valued).</p> <p>This attribute is used to define the types of events an application is privileged to send to the provisioning integration service. The mapping takes this form:</p> <p><i>Event_Object: affected_domain:operation(attributes, . . .)</i></p> <p>In the mapping IDENTITY:cn=users,dc=acme,dc=com:ADD(*) the IDENTITY_ADD event is allowed for the specified domain and all attributes are also allowed. In the mapping IDENTITY:cn=users,dc=acme,dc=com:MODIFY(cn,sn.mail,telephonenumber), the IDENTITY_MODIFY event is allowed only for the attributes in the list. Any extra attributes are silently ignored.</p>

PL/SQL Bidirectional Interface (Version 3.0)

Before attempting to use Version 3.0 of the PL/SQL interface, please refer to:

- [Appendix A, "Java Plug-ins for User Provisioning"](#)
- The Oracle Provisioning Service Concepts chapter in *Oracle Identity Management Integration Guide*
- The Deploying Provisioning-Integrated Applications chapter in *Oracle Identity Management Integration Guide*

The PL/SQL callback interface requires you to develop a PL/SQL package that Oracle Directory Provisioning Integration Service invokes in the application specific database. Choose any name for the package, but be sure to use the same name when you register the package at subscription time. Implement the package by using the following PL/SQL package specification:

```
DROP TYPE LDAP_EVENT_LIST_V3;
DROP TYPE LDAP_EVENT_V3;
DROP TYPE LDAP_EVENT_STATUS_LIST_V3;
DROP TYPE LDAP_ATTR_LIST_V3;
DROP TYPE LDAP_ATTR_V3;
```

```

DROP TYPE LDAP_ATTR_VALUE_LIST_V3;
DROP TYPE LDAP_ATTR_VALUE_V3;
-----
-----
-- Name: LDAP_ATTR_VALUE_V3
-- Data Type: OBJECT
-- DESCRIPTION: This structure contains values of an attribute. A list of one or
more of this object is passed in any event.
-----
-----

CREATE TYPE LDAP_ATTR_VALUES_V3 AS OBJECT (
    attr_value      VARCHAR2(4000),
    attr_bvalue     RAW(2048),
    attr_value_len  INTEGER,
);

GRANT EXECUTE ON LDAP_ATTR_VALUE_V3 to public;

CREATE TYPE LDAP_ATTR_VALUE_LIST_V3 AS TABLE OF LDAP_ATTR_VALUE_V3;
/
GRANT EXECUTE ON LDAP_ATTR_VALUE_LIST_V3 to public;
-----
-----
-- Name: LDAP_ATTR_V3
-- Data Type: OBJECT
-- DESCRIPTION: This structure contains details regarding an attribute. A list of
one or more of this object is passed in any event.
-----
-----

CREATE TYPE LDAP_ATTR_V3 AS OBJECT (
    attr_name       VARCHAR2(256),
    attr_type       INTEGER ,
    attr_mod_op     INTEGER,
    attr_values     LDAP_ATTR_VALUE_LIST
);

GRANT EXECUTE ON LDAP_ATTR_V3 to public;

CREATE TYPE LDAP_ATTR_LIST_V3 AS TABLE OF LDAP_ATTR_V3;
/
GRANT EXECUTE ON LDAP_ATTR_LIST_V3 to public;
-----
-----
-- Name: LDAP_EVENT_V3
-- Data Type: OBJECT
-- DESCRIPTION: This structure contains event information plus the attribute List.
-----
-----

CREATE TYPE LDAP_EVENT_V3 AS OBJECT (
    event_type  VARCHAR2(32),
    event_id   VARCHAR2(32),
    event_src  VARCHAR2(1024),
    event_time VARCHAR2(32),
    object_name VARCHAR2(1024),
    object_type VARCHAR2(32),
    object_guid VARCHAR2(32),
    object_dn  VARCHAR2(1024),
    profile_id VARCHAR2(1024),

```

```

        attr_list    LDAP_ATTR_LIST_V3 ) ;
/

GRANT EXECUTE ON LDAP_EVENT_V3 to public;
CREATE TYPE LDAP_EVENT_LIST_V3 AS TABLE OF LDAP_EVENT_V3;
/
GRANT EXECUTE ON LDAP_EVENT_LIST_V3 to public;
-----
-----
-- Name: LDAP_EVENT_STATUS_V3
-- Data Type: OBJECT
-- DESCRIPTION: This structure contains information that is sent by the consumer
of an event to the supplier in response to the actual event.
-----
-----

CREATE TYPE LDAP_EVENT_STATUS_V3 AS OBJECT (
    event_id    VARCHAR2(32),
    status      VARCHAR2(32),
    status_msg  VARCHAR2(2048),
    object_guid VARCHAR(32),
) ;
/

GRANT EXECUTE ON LDAP_EVENT_STATUS_V3 to public;
CREATE TYPE LDAP_EVENT_STATUS_LIST_V3 AS TABLE OF LDAP_EVENT_STATUS_V3;
/
GRANT EXECUTE ON LDAP_EVENT_STATUS_LIST_V3 to public;
-----
-----
-- Name: LDAP_NTIFY
-- DESCRIPTION: This is the interface to be implemented by provisioning integrated
applications to send information to and receive information from the directory.
The name of the package can be customized as needed. The function and procedure
names within this package should not be changed.
-----
-----

CREATE OR REPLACE PACKAGE LDAP_NTIFY AS

    -- The Predefined Event Types

    ENTRY_ADD      CONSTANT VARCHAR2 (32) := 'ENTRY_ADD' ;
    ENTRY_DELETE   CONSTANT VARCHAR2 (32) := 'ENTRY_DELETE' ;
    ENTRY_MODIFY   CONSTANT VARCHAR2 (32) := 'ENTRY_MODIFY' ;

    USER_ADD       CONSTANT VARCHAR2 (32) := 'USER_ADD' ;
    USER_DELETE    CONSTANT VARCHAR2 (32) := 'USER_DELETE' ;
    USER_MODIFY    CONSTANT VARCHAR2 (32) := 'USER_MODIFY' ;

    IDENTITY_ADD   CONSTANT VARCHAR2 (32) := 'IDENTITY_ADD' ;
    IDENTITY_DELETE CONSTANT VARCHAR2 (32) := 'IDENTITY_DELETE' ;
    IDENTITY_MODIFY CONSTANT VARCHAR2 (32) := 'IDENTITY_MODIFY' ;

    GROUP_ADD      CONSTANT VARCHAR2 (32) := 'GROUP_ADD' ;
    GROUP_DELETE   CONSTANT VARCHAR2 (32) := 'GROUP_DELETE' ;
    GROUP_MODIFY   CONSTANT VARCHAR2 (32) := 'GROUP_MODIFY' ;

```



```

SUBSCRIPTION_ADD      CONSTANT VARCHAR2(32) := 'SUBSCRIPTION_ADD';
SUBSCRIPTION_DELETE   CONSTANT VARCHAR2(32) := 'SUBSCRIPTION_DELETE';
SUBSCRIPTION_MODIFY   CONSTANT VARCHAR2(32) := 'SUBSCRIPTION_MODIFY';

SUBSCRIBER_ADD        CONSTANT VARCHAR2(32) := 'SUBSCRIBER_ADD';
SUBSCRIBER_DELETE     CONSTANT VARCHAR2(32) := 'SUBSCRIBER_DELETE';
SUBSCRIBER_MODIFY     CONSTANT VARCHAR2(32) := 'SUBSCRIBER_MODIFY';

-- The Attribute Type

ATTR_TYPE_STRING      CONSTANT NUMBER := 0;
ATTR_TYPE_BINARY      CONSTANT NUMBER := 1;
ATTR_TYPE_ENCRYPTED_STRING CONSTANT NUMBER := 2;

-- The Attribute Modification Type

MOD_ADD               CONSTANT NUMBER := 0;
MOD_DELETE            CONSTANT NUMBER := 1;
MOD_REPLACE           CONSTANT NUMBER := 2;

-- The Event dispositions constants

EVENT_SUCCESS         CONSTANT VARCHAR2(32) := 'EVENT_SUCCESS';
EVENT_IN_PROGRESS     CONSTANT VARCHAR2(32) := 'EVENT_IN_PROGRESS';
EVENT_USER_NOT_REQUIRED CONSTANT VARCHAR2(32) := 'EVENT_USER_NOT_REQUIRED';
EVENT_ERROR           CONSTANT VARCHAR2(32) := 'EVENT_ERROR';
EVENT_ERROR_ALERT     CONSTANT VARCHAR2(32) := 'EVENT_ERROR_ALERT';
EVENT_ERROR_ABORT     CONSTANT VARCHAR2(32) := 'EVENT_ERROR_ABORT';

-- The Actual Callbacks

FUNCTION GetAppEvents (events OUT LDAP_EVENT_LIST_V3)
RETURN NUMBER;

-- Return CONSTANTS
EVENT_FOUND           CONSTANT NUMBER := 0;
EVENT_NOT_FOUND       CONSTANT NUMBER := 1403;

```

If the provisioning server is unable to process an inbound event, it triggers an `EVENT_ERROR_ALERT` status, which generates a trigger in Oracle Enterprise Manager.

If the provisioning server is able to process the event, but finds that the event cannot be processed—for example, the user to be modified, subscribed, or deleted does not exist—it responds with `EVENT_ERROR` to indicate to the application that something is wrong. It is again up to the application to handle the status event.

`EVENT_ERROR` means no errors in directory operations. The event cannot be processed for other reasons.

`PutAppEventStatus()` : DIP Server invokes this callback in the remote Data base after processing an event it had received using the `GetAppEvents()` callback. For every event received, the DIP server sends the status event back after processing the event. This API will NOT be required by the Oracle Collaboration Suite release 3.0 components.

```
PROCEDURE PutAppEventStatus (event_status IN LDAP_EVENT_STATUS_LIST_V3);
```

`PutOIDEvents()` : DIP Server invokes this API in the remote Database. DIP server sends event to applications using this callback. It also expects a status event object in response as an OUT parameter. This API needs to be implemented by all the Oracle Collaboration Suite release 3.0 components.

```
PROCEDURE PutOIDEvents (event          IN LDAP_EVENT_LIST_V3,
                        event_status OUT LDAP_EVENT_STATUS_LIST_V3);

END LDAP_NTIFY;
/
```

PL/SQL Bidirectional Interface (Version 2.0)

The PL/SQL callback interface requires that you develop a PL/SQL package that the provisioning integration service invokes in the application-specific database. Choose any name for the package, but be sure to use the same name when you register the package at subscription time. Implement the package using the following PL/SQL package specification:

```
DROP TYPE LDAP_EVENT;
DROP TYPE LDAP_EVENT_STATUS;
DROP TYPE LDAP_ATTR_LIST;
DROP TYPE LDAP_ATTR;
-----
-- Name: LDAP_ATTR
-- Data Type: OBJECT

DESCRIPTION: This structure contains details regarding an attribute. A list of one
--           or more of this object is passed in any event.
-----

CREATE TYPE LDAP_ATTR AS OBJECT (
    attr_name      VARCHAR2(256),
    attr_value     VARCHAR2(4000),
    attr_bvalue    RAW(2048),
    attr_value_len INTEGER,
    attr_type      INTEGER ,
    attr_mod_op    INTEGER
);

GRANT EXECUTE ON LDAP_ATTR to public;

CREATE TYPE LDAP_ATTR_LIST AS TABLE OF LDAP_ATTR;
/
GRANT EXECUTE ON LDAP_ATTR_LIST to public;

-----
-- Name: LDAP_EVENT
-- Data Type: OBJECT
-- DESCRIPTION: This structure contains event information plus the attribute
--             list.
-----

CREATE TYPE LDAP_EVENT AS OBJECT (
    event_type VARCHAR2(32),
    event_id   VARCHAR2(32),
    event_src  VARCHAR2(1024),
    event_time VARCHAR2(32),
    object_name VARCHAR2(1024),
    object_type VARCHAR2(32),
    object_guid VARCHAR2(32),
    object_dn  VARCHAR2(1024),
```

```

        profile_id VARCHAR2(1024),
        attr_list  LDAP_ATTR_LIST ) ;
/

GRANT EXECUTE ON LDAP_EVENT to public;

-----
-----
-- Name: LDAP_EVENT_STATUS
-- Data Type: OBJECT
-- DESCRIPTION: This structure contains information that is sent by the
--               consumer of an event to the supplier in response to the
--               actual event.
-----
-----

CREATE TYPE LDAP_EVENT_STATUS AS OBJECT (
        event_id          VARCHAR2(32),
        orclguid          VARCHAR(32),
        error_code        INTEGER,
        error_String      VARCHAR2(1024),
        error_disposition VARCHAR2(32)) ;
/

GRANT EXECUTE ON LDAP_EVENT_STATUS to public;

```

Provisioning Event Interface (Version 1.1)

You must develop logic to consume events generated by the provisioning integration service. The interface between the application and the provisioning integration service can be table-based, or it can use PL/SQL callbacks.

The PL/SQL callback interface requires that you develop a PL/SQL package that the provisioning integration service invokes in the application-specific database. Choose any name for the package, but be sure to use the same name when you register the package at subscription time. Implement the package using the following PL/SQL package specification:

```

Rem
Rem      NAME
Rem      ldap_ntfy.pks - Provisioning Notification Package Specification.
Rem

DROP TYPE LDAP_ATTR_LIST;
DROP TYPE LDAP_ATTR;

-- LDAP ATTR
-----
--
-- Name          : LDAP_ATTR
-- Data Type     : OBJECT
-- DESCRIPTION   : This structure contains details regarding
--               an attribute.
--
-----

CREATE TYPE LDAP_ATTR AS OBJECT (
        attr_name        VARCHAR2(255),
        attr_value       VARCHAR2(2048),
        attr_bvalue      RAW(2048),

```

```

        attr_value_len    INTEGER,
        attr_type         INTEGER -- (0 - String, 1 - Binary)
        attr_mod_op      INTEGER
    );
/
GRANT EXECUTE ON LDAP_ATTR to public;

-----
--
-- Name          : LDAP_ATTR_LIST
-- Data Type     : COLLECTION
-- DESCRIPTION   : This structure contains collection
--                 of attributes.
--
-----

CREATE TYPE LDAP_ATTR_LIST AS TABLE OF LDAP_ATTR;
/
GRANT EXECUTE ON LDAP_ATTR_LIST to public;

-----
--
-- NAME          : LDAP_NTIFY
-- DESCRIPTION   : This is a notifier interface implemented by Provisioning System
--                 clients to receive information about changes in Oracle Internet
--                 Directory. The name of package can be customized as needed.
--                 The function names within this package should not be changed.
--
-----

CREATE OR REPLACE PACKAGE LDAP_NTIFY AS

--
-- LDAP_NTIFY data type definitions
--

-- Event Types
USER_DELETE      CONSTANT VARCHAR2(256) := 'USER_DELETE';
USER_MODIFY      CONSTANT VARCHAR2(256) := 'USER_MODIFY';
GROUP_DELETE     CONSTANT VARCHAR2(256) := 'GROUP_DELETE';
GROUP_MODIFY     CONSTANT VARCHAR2(256) := 'GROUP_MODIFY';

-- Return Codes (Boolean)
SUCCESS          CONSTANT NUMBER := 1;
FAILURE          CONSTANT NUMBER := 0;

-- Values for attr_mod_op in LDAP_ATTR object.
MOD_ADD          CONSTANT NUMBER := 0;
MOD_DELETE       CONSTANT NUMBER := 1;
MOD_REPLACE      CONSTANT NUMBER := 2;

-----
-- Name: LDAP_NTIFY
-- DESCRIPTION: This is the interface to be implemented by Provisioning System
--                 clients to send information to and receive information from
--                 Oracle Internet Directory. The name of the package can be
--                 customized as needed. The function names within this package
--                 should not be changed.
-----

```

```
-----
CREATE OR REPLACE PACKAGE LDAP_NTIFY AS
```

Predefined Event Types

```
ENTRY_ADD          CONSTANT VARCHAR2 (32)  := 'ENTRY_ADD';
ENTRY_DELETE       CONSTANT VARCHAR2 (32)  := 'ENTRY_DELETE';
ENTRY_MODIFY       CONSTANT VARCHAR2 (32)  := 'ENTRY_MODIFY';

USER_ADD          CONSTANT VARCHAR2 (32)  := 'USER_ADD';
USER_DELETE       CONSTANT VARCHAR2 (32)  := 'USER_DELETE';
USER_MODIFY CONSTANT VARCHAR2(32)        := 'USER_MODIFY';

IDENTITY_ADD      CONSTANT VARCHAR2 (32)  := 'IDENTITY_ADD';
IDENTITY_DELETE   CONSTANT VARCHAR2 (32)  := 'IDENTITY_DELETE';
IDENTITY_MODIFY   CONSTANT VARCHAR2 (32)  := 'IDENTITY_MODIFY';

GROUP_ADD        CONSTANT VARCHAR2 (32)  := 'GROUP_ADD';
GROUP_DELETE     CONSTANT VARCHAR2 (32)  := 'GROUP_DELETE';
GROUP_MODIFY     CONSTANT VARCHAR2 (32)  := 'GROUP_MODIFY';

SUBSCRIPTION_ADD  CONSTANT VARCHAR2(32)    := 'SUBSCRIPTION_ADD';
SUBSCRIPTION_DELETE CONSTANT VARCHAR2(32) := 'SUBSCRIPTION_DELETE';
SUBSCRIPTION_MODI CONSTANT VARCHAR2(32)    := 'SUBSCRIPTION_MODIFY';

SUBSCRIBER_ADD   CONSTANT VARCHAR2(32)    := 'SUBSCRIBER_ADD';
SUBSCRIBER_DELETE CONSTANT VARCHAR2(32)    := 'SUBSCRIBER_DELETE';
SUBSCRIBER_MODIFY CONSTANT VARCHAR2(32)    := 'SUBSCRIBER_MODIFY';
```

Attribute Type

```
ATTR_TYPE_STRING      CONSTANT NUMBER  := 0;
ATTR_TYPE_BINARY      CONSTANT NUMBER  := 1;
ATTR_TYPE_ENCRYPTED_STRING CONSTANT NUMBER  := 2;
```

Attribute Modification Type

```
MOD_ADD      CONSTANT NUMBER  := 0;
MOD_DELETE   CONSTANT NUMBER  := 1;
MOD_REPLACE  CONSTANT NUMBER  := 2;
```

Event Dispositions Constants

```
EVENT_SUCCESS  CONSTANT VARCHAR2(32) := 'EVENT_SUCCESS';
EVENT_ERROR    CONSTANT VARCHAR2(32) := 'EVENT_ERROR';
EVENT_RESEND   CONSTANT VARCHAR2(32) := 'EVENT_RESEND';
```

Callbacks

A callback is a function invoked by the provisioning integration service to send or receive notification events. While transferring events for an object, the related attributes can also be sent along with other details. The attributes are delivered as a collection (array) of attribute containers, which are in unnormalized form: if an attribute has two values, two rows are sent in the collection.

GetAppEvent()

The directory integration and provisioning server invokes this API in the remote database. It is up to the application to respond with an event. The Oracle Directory Integration and Provisioning platform processes the event and sends the status back using the `PutAppEventStatus()` callback. The return value of `GetAppEvent()` indicates whether an event is returned or not.

```
FUNCTION GetAppEvent (event OUT LDAP_EVENT)
RETURN NUMBER;

-- Return CONSTANTS
EVENT_FOUND          CONSTANT NUMBER := 0;
EVENT_NOT_FOUND      CONSTANT NUMBER := 1403;
```

If the provisioning server is not able to process the event—that is, it runs into some type of LDAP error—it responds with `EVENT_RESEND`. The application is expected to resend that event when `GetAppEvent()` is invoked again.

If the provisioning server is able to process the event, but finds that the event cannot be processed—for example, the user to be modified does not exist, or the user to be subscribed does not exist, or the user to be deleted does not exist—then it responds with `EVENT_ERROR` to indicate to the application that something was wrong. Resending the event is not required. It is up to the application to handle the event.

Note the difference between `EVENT_RESEND` and `EVENT_ERROR` in the previous discussion. `EVENT_RESEND` means that it was possible to apply the event but the server could not. If it gets the event again, it might succeed.

`EVENT_ERROR` means there is no error in performing directory operations, but the event could not be processed due to other reasons.

PutAppEventStatus()

The directory integration and provisioning server invokes this callback in the remote database after processing an event it has received using the `GetAppEvent()` callback. For every event received, the directory integration and provisioning server sends the status event back after processing the event.

```
PROCEDURE PutAppEventStatus (event_status IN LDAP_EVENT_STATUS);
```

PutOIDEvent()

The directory integration and provisioning server invokes this API in the remote database. It sends event to applications using this callback. It also expects a status event object in response as an `OUT` parameter. If a valid event status object is not sent back, or it indicates a `RESEND`, the directory integration and provisioning server resends the event. In case of `EVENT_ERROR`, the server does not resend the event.

```
PROCEDURE PutOIDEvent (event IN LDAP_EVENT, event_status OUT LDAP_EVENT_
STATUS);
END LDAP_NTIFY;
/
```

Part III

Appendixes

Part III presents plug-ins that can be used to customize provisioning in Oracle Collaboration Suite. In addition, this section contains an appendix about DSML syntax and usage.

- [Appendix A, "Java Plug-ins for User Provisioning"](#)
- [Appendix B, "DSML Syntax"](#)

Java Plug-ins for User Provisioning

This appendix explains how to use plug-ins to customize provisioning policy evaluation, data validation, data manipulation, and event delivery in typical deployments of Oracle Provisioning Service version 3.0.

The Oracle provisioning server cannot support all of the provisioning needs of a deployment. Hence, hooks are provided at various stages of user creation, modification, and deletion. These hooks enable an enterprise to incorporate its own business rules and to tailor information creation to its needs. The hooks take the form of Java plug-ins.

This appendix contains these topics:

- [Plug-in Types and Their Purpose](#)
- [Plug-in Requirements](#)
- [Data Entry Plug-in](#)
- [Data Access Plug-in](#)
- [Event Delivery Plug-in](#)
- [Plug-in Return Status](#)
- [Configuration Template](#)
- [Sample Code](#)

Plug-in Types and Their Purpose

There are three types of plug-ins:

- Data entry plug-ins
- Data manipulation and data access plug-ins
- Event Delivery plug-ins

The data entry plug-ins can be used by applications that integrate with the provisioning framework using either synchronous or asynchronous provisioning. The data access plug-ins are used only by applications that are integrated with the provisioning framework for synchronous provisioning. The event delivery plug-ins are used only by applications that integrate with the provisioning framework using asynchronous provisioning.

Oracle Provisioning Console, Oracle integration and provisioning server, and other mechanisms that affect the base user information in the directory invoke these plug-ins when the information is created. By configuring a data entry plug-in, a deployment can do any of the following:

- Validate attribute values for application users
- Validate attribute values for base users
- Enhance attribute values for application users
- Enhance attribute values for base users
- Evaluate provisioning policies

If you want the deployed application to maintain application user information you must configure a data access plug-in for it. This type of plug-in enables you to maintain the application information either outside of the directory or within it as several entries.

Data entry and data access plug-ins are typically invoked from one of these environments:

- User provisioning console for Oracle Delegated Administration Services
- Oracle directory integration and provisioning server
- Provisioning API
- Bulk Provisioning Tools

The event delivery plug-ins are required by applications that have the JAVA interface type and that subscribe for provisioning events. Applications that have synchronous provisioning should not implement event delivery plug-ins.

Plug-in Requirements

All of the plug-ins that you provide for an application must be in a JAR file that can be uploaded to the directory with the standard LDIF template. See the section "[Configuration Template](#)" for an example. The plug-in interface definitions are found in `$ORACLE_HOME/jlib/ldapjclnt10.jar`. Refer to *Oracle Internet Directory API Reference* and the public interfaces for a more detailed description. If the application requires additional jar files, you can upload them too.

Data Entry Plug-in

Data entry plug-ins take two forms:

- Pre-data-entry plug-ins
- Post-data-entry plug-ins

If you want to use either of these plug-ins, you must implement the `oracle.idm.provisioning.plugin.IdataEntryPlugin` interface. This interface has three methods. Here it is:

```
/**
 * The applications can perform a post data entry operation by
 * implementing this method.
 *
 * @param appCtx the application context
 * @param idmUser the IdmUser object
 * @param baseUserAttr Base user properties
 * @param appUserAttr App user properties
 * @throws PluginException when an exception occurs.
 */
public PluginStatus process(ApplicationContext appCtx,
    IdmUser idmUser, ModPropertySet baseUserAttr,
```

```

        ModPropertySet appUserAttr)throws PluginException;
/**
 * Returns the Modified Base User properties
 *
 * @return ModPropertySet modified base user properties.
 */
public ModPropertySet getBaseAttrMods();

/**
 * Returns the Modified App User properties
 *
 * @return ModPropertySet modified app user properties.
 */
public ModPropertySet getAppAttrMods();

```

Typically the plug-in implementer uses these methods for data validation or policy evaluation. In the latter case, a base user attribute is used to make the decision.

The application context object contains this information:

- **LDAP directory context**

If you want the application to perform a directory operation, you can have it obtain the LDAP context from the application object. Note that this LDAP context should not be closed in the plug-in.
- **Plug-in call mode**

The plug-in is called from Oracle Provisioning Console, Oracle directory integration and provisioning server, or another environment that invokes the provisioning API. If the calling environment is Oracle Directory Integration and Provisioning, the provisioning service calls the plug-in. The two possible values are `INTERACTIVE_MODE` and `AUTOMATIC_MODE`. The first indicates that the plug-in was invoked through interaction between Oracle Delegated Administration Services and a client application. The second indicates that the plug-in was invoked by Oracle Directory Integration and Provisioning, where user intervention does not occur.
- **Client locale**

The plug-in may want to know what the client locale is, especially if it is invoked from Oracle Delegated Administration Services.
- **Plug-in call operation**

You may decide to have data entry plug-ins for both create and modify user operations. You may even implement these plug-ins in the same class. Under these conditions, the plug-in must determine which operation is invoked. The application context object uses the values `OP_CREATE` and `OP_MODIFY` to identify the operation.
- **Plug-in invocation point**

The data entry plug-in is typically used to determine whether a user needs to be provisioned for an application. The policy evaluation and data validation that occurs can be performed in either a pre-data-entry plug-in or a post-data-entry plug-in. You may choose either or both. If you choose both, you can implement them in the same class. The application context object specifies which one is actually invoked. It uses the values `PRE_DATA_ENTRY` and `POST_DATA_ENTY` to do this.
- **Callback context**

If you decide to have both pre and post plug-ins for an operation and you want the pre plug-in to share information with the post plug-in, you can set the callback context in the application context object of the pre-data-entry plug-in. The post-data-entry plug-in can then obtain and use this callback context.

- **Logging**

You can use the log methods provided in the application context object to log information for the plug-in.

The calling sequence looks like this:

1. Download and instantiate a plug-in object based on the configuration information object in Oracle Internet Directory
2. Construct an application context object that will be passed to the plug-in.
3. Call `process method()`
4. Call `getBaseAttrMods()` to obtain base user attributes that are modified in `process()`.
5. Merge the base user attributes returned by `getBaseAttrMods()` with the base user attributes, depending on the plug-in execution status. The execution status can be either `success` or `failure`. The plug-in implementer must return a valid plug-in execution status object. If null is returned, the execution status is considered a failure.
6. Merging of the base user will only be done if the plug-in execution status is successful.
7. Call `getAppAttrMods()` for the plug-in. This method obtains application user attributes that are modified in `process()`.
8. Merge the application user attributes returned by `getAppAttrMods()` with the application user attributes, depending on the user provisioning status returned by the plug-in.

Pre-Data-Entry Plug-in

The pre-data-entry plug-in generates values for application attributes. The attribute defaults specified during application registration are passed to this plug in along with the current base user attributes. The returned values are displayed in the UI if the invocation environment is interactive like Oracle Delegated Administration Services.

The pre-data-entry plug-in can decide whether the user should be provisioned for an application. The plug-in examines base user attributes to make the decision. It is invoked during create and modify operations. You can support both operations with one plug-in class, or you can assign one class to each.

If the application decides to have pre-data- entry plug-ins for create and modify operations, two configuration entries must be created in Oracle Internet Directory under the application container. The first entry is for the create operation:

This

```
dn: cn=PRE_DATA_ENTRY_CREATE, cn=Plugins, cn=FILES, cn=Applications,
cn=Provisioning, cn=Directory Integration Platform, cn=Products,
cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: oracle.myapp.provisioning.UserCreatePlugin
```

```
orclODIPPluginAddInfo: Pre Data Entry Plugin for CREATE operation
```

The second entry is for the modify operation:

```
dn: cn=PRE_DATA_ENTRY_MODIFY, cn=Plugins, cn=FILES, cn=Applications,
   cn=Provisioning, cn=Directory Integration Platform, cn=Products,
   cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: oracle.myapp.provisioning.UserModifyPlugin
orclODIPPluginAddInfo: Pre Data Entry Plugin for MODIFY operation
```

In this example, separate classes for create and modify plug-ins are shown.

Post-Data-Entry Plug-in

The post-data-entry plug-in validates data entered by the user in the UI. In addition, it generates derived attribute values. If the plug in fails for any one application, the UI does not proceed. All applications must successfully validate the data before a user entry can be created in the directory. However, in the case of non-UI environment or automatic route, the plug-in implementer can decide to raise an error or continue, based on the plug-in call mode (`INTERACTIVE_MODE` or `AUTOMATIC_MODE`).

Like the pre-data-entry plug-in, the post-data-entry plug-in is invoked during create and modify operations. The application can decide to implement one plug-in class for both operations or a separate class for each.

If you decide to have post-data-entry plug-ins for create and modify operations, create two configuration entries in Oracle Internet Directory under the application container. The first entry is for the create operation:

```
dn: cn=POST_DATA_ENTRY_CREATE, cn=Plugins, cn=FILES, cn=Applications,
   cn=Provisioning, cn=Directory Integration Platform, cn=Products,
   cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: oracle.myapp.provisioning.UserMgmtPlugin
orclODIPPluginAddInfo: Post Data Entry Plugin for CREATE and MODIFY
operations
```

The second entry is for the modify operation:

```
dn: cn=POST_DATA_ENTRY_MODIFY, cn=Plugins, cn=FILES, cn=Applications,
   cn=Provisioning, cn=Directory Integration Platform, cn=Products,
   cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: oracle.myapp.provisioning.UserMgmtPlugin
orclODIPPluginAddInfo: Post Data Entry Plugin for MODIFY and CREATE operation
```

In this example, too, separate classes for create and modify plug-ins are shown.

Data Access Plug-in

The primary purpose of the data access plug in is to manage the application-specific information of the user in the directory. You can use this plug-in to create and retrieve the information.

The data access plug-in is invoked whenever a user is created and is requesting provisioning for an application—whether by Oracle Delegated Administration Services, by Oracle Directory Integration and Provisioning, or by bulk provisioning tools.

The data access plug-in is invoked during modify and delete operations as well. It can update the application information or remove it.

If you want to use the data access plug-in, implement the interface `oracle.idm.provisioning.plugin.IDataAccessPlugin`. Here is the interface:

```
/**
 * The applications can create/modify/delete the user footprint by
 * implementing this method.
 *
 * @param appCtx the application context
 * @param idmUser IdmUser object
 * @param baseUserAttr Base user properties
 * @param appUserAttr App user properties
 *
 * @return PluginStatus a plugin status object, which must contain
 * the either <code>IdmUser.PROVISION_SUCCESS</CODE> or
 * <code>IdmUser.PROVISION_FAILURE</CODE> provisioning status
 *
 * @throws PluginException when an exception occurs.
 */
public PluginStatus process(ApplicationContext appCtx,
    IdmUser idmUser, ModPropertySet baseUserAttr,
    ModPropertySet ppUserAttr) throws PluginException;

/**
 * The applications can return their user footprint by
 * implementing this method. Use <CODE>
 * oracle.ldap.util.VarPropertySet </CODE>
 * as the return object
 *
 * <PRE>
 * For Ex.
 *   PropertySet retPropertySet = null;
 *   retPropertySet = new VarPropertySet();
 *
 * //Fetch the App data and add it to retPropertySet
 *   retPropertySet.addProperty("name", "value");
 *   ..
 *   return retPropertySet;
 * </PRE>
 *
 * @throws PluginException when an exception occurs.
 */
public PropertySet getAppUserData(ApplicationContext appCtx,
    IdmUser user, String reqAttrs[]) throws PluginException;
```

If you want to manage the user information for an application, create a plug-in configuration entry in the directory under the application container. The example that follows shows what this entry looks like:

```
dn: cn=DATA_ACCESS, cn=Plugins, cn=FILES, cn=Applications,
    cn=Provisioning, cn=Directory Integration Platform, cn=Products,
    cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
```

```

orclStatus: ENABLE
orclODIPPluginExecName: oracle.myapp.provisioning.UserDataAccPlugin
orclODIPPluginAddInfo: Data Access Plugin

```

Event Delivery Plug-in

The primary purpose of the event delivery plug-in is to use the events notified by the Oracle integration and provisioning server. Events are delivered to the plug-in by the Oracle integration and provisioning server. Based on the event type and the action to be performed in the application repository, the plug-in performs the required operations. The interface definitions for this plug-in are as follows:

```

/* $Header: IEventPlugin.java 09-jun-2005.12:45:53 *
/* Copyright (c) 2004, 2005, Oracle. All rights reserved. */
/*
DESCRIPTION
All of the plug-in interfaces must extend this common interface.
PRIVATE CLASSES
None
NOTES
None
*/
package oracle.idm.provisioning.plugin;
/**
 * This is the base interface
 */
public interface IEventPlugin
{
/**
 * The applications can perform the initialization logic in this method.
 *
 * @param Object For now it is the provisioning Profile that will be passed.
 *         look at oracle.ldap.odip.engine.ProvProfile for more details.
 *
 * @throws PluginException when an exception occurs.
 */
public void initialize(Object profile) throws PluginException;
/**
 * The applications can perform the termination logic in this method.
 *
 * @param void Provisioning Profile Object will be sent.
 *         refer to oracle.ldap.odip.engine.ProvProfile for more details
 * @throws PluginException when an exception occurs.
 */
public void terminate(Object profile) throws PluginException;
/**
 * Set Additional Info.
 * Since we pass on the complete profile, there is no requirement to set
 * the additiona
 * @param addInfo Plugin additional info
 */
//public void setAddInfo(Object addInfo);
}

/* $Header: IEventsFromOID.java 09-jun-2005.12:45:53 *
/* Copyright (c) 2004, 2005, Oracle. All rights reserved. */
/*
DESCRIPTION
Applications interested in receiving changes from OID should

```

```

implement this
    interface.
    PRIVATE CLASSES
        <None>
    NOTES
*/
package oracle.idm.provisioning.plugin;
import oracle.idm.provisioning.event.Event;
import oracle.idm.provisioning.event.EventStatus;

/**
 * Applications interested in receiving changes from OID should implement this
 * interface. The applications register with the OID for the changes occurring
 * at OID. The DIP engine would instantiate an object of this class and invoke
 * the initialize(), sendEventsToApp(), and truncate() method in the same
 * sequence. The initialize method would provide the appropriate information
 * from the profile in the form of a java.util.Hashtable object.
 * The property names, i.e. the hash table key that could be used by the
 * interface implementer will be defined as constants in this interface.
 *
 * @version $Header: IEventsFromOID.java 09-jun-2005.12:45:53 $
 */
public interface IEventsFromOID extends IEventPlugin
{

    /**
     * Initialize. The application would provide any initialization logic
     * through method. The DIP engine after instantiating a class that
     * implements this interface will first invoke this method.
     *
     * @param prop A HashMap that would contain necessary information exposed
     * to the applications
     * @throws EventInitializationException the applications must throw this
     * exception in case of error.
     */
    public void initialize(Object provProfile)
        throws EventPluginInitException;

    /**
     * OID Events are delivered to the application through this method.
     *
     * @param evts an array of LDATEvent objects returned by the DIP engine
     * @return the application logic must process these events and return the
     * status of the processed events
     * @throws EventDeliveryException the applications must throw this exception
     * in case of any error.
     */
    public EventStatus[] sendEventsToApp(Event [] evts)
        throws EventDeliveryException;
}

/* $Header: IEventsToOID.java 09-jun-2005.12:45:53 $ */
/* Copyright (c) 2004, 2005, Oracle. All rights reserved. */

/*
    DESCRIPTION
    Applications interested in sending changes to OID should implement this
    interface.
*/
package oracle.idm.provisioning.plugin;

```



```

import oracle.idm.provisioning.event.Event;
import oracle.idm.provisioning.event.EventStatus;

/**
 * Applications interested in sending changes to OID should implement this
 * interface. The applications must register with the OID for the sending
 * changes at their end to DIP. The DIP engine would instantiate an object
 * of this class and invoke the initialize(), sendEventsFromApp(), and
 * truncate() method in the same sequence. The initialize method would
 * provide the appropriate information from the profile in the form of
 * a java.util.Hashtable object. The property names, i.e. the hash table key
 * that could be used by the interface implementer will be defined as
 * constants in this interface.
 *
 */
public interface IEventsToOID extends IEventPlugin
{
    /**
     * Initialize. The application would provide any initialization logic
     * through method. The DIP engine after instantiating a class that
     * implements this interface will first invoke this method.
     *
     * @param prop ProvProfile
     *         oracle.ldap.odip.engine.ProvProfile
     * @throws EventPluginInitException the applications must throw this
     *         exception in case of error.
     */
    public void initialize(Object profile) throws EventPluginInitException;

    /**
     * Application Events are delivered to OID through this method.
     *
     * @return an array of Event objects returned to be processed by the
     *         DIP engine.
     * @throws EventDeliveryException the applications must throw this exception
     *         in case of any error.
     */
    public Event[] receiveEventsFromApp()
        throws EventDeliveryException;

    /**
     * Application can let the DIP engine know whether there are more event to
     * follow through this method
     *
     * @return true if there are more events to be returned and false otherwise
     * @throws PluginException the applications must throw this exception
     *         in case of any error.
     */
    public boolean hasMore() throws PluginException;

    /**
     * The status of the application events are intimated through this method.
     * i.e the DIP engine after processing the events calls this method to set
     * the event status.
     *
     * @param an array of Event status objects describing the processed event
     *         status by the DIP engine.
     * @throws EventDeliveryException the applications must throw this exception
     *         in case of any error.
     */
    public void setAppEventStatus(EventStatus[] evtStatus)

```

```

        throws EventDeliveryException;
    }

```

Plug-in Return Status

Each of the provisioning plug-ins must return an object of the class `oracle.idm.provisioning.plugin.PluginStatus`. This object indicates the execution status, which is either success or failure. The object can return the user provisioning status as well.

Configuration Template

The LDIF template provided here is used in Oracle Internet Directory 10g Release 2 (10.1.2) to specify the application plug-in. You must create a directory entry for the application and upload the JAR file that contains the classes that implement the plug-in.

```

dn: cn=Plugins, cn=APPTYPE, cn=Applications, cn=Provisioning,
   cn=Directory Integration Platform, cn=Products, cn=OracleContext
changetype: add
add: orclODIPPluginExecData
orclODIPPluginExecData: full_path_name_of_the_JAR_file
objectclass: orclODIPPluginContainer

```

```

dn: cn=PRE_DATA_ENTRY_CREATE, cn=Plugins, cn=APPTYPE, cn=Applications,
   cn=Provisioning, cn=Directory Integration Platform, cn=Products,
   cn=OracleContext
   cn=Provisioning, cn=Directory Integration Platform, cn=Products,
   cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: Name_of_the_class_that_implements_the_plug-in
orclODIPPluginAddInfo: Pre Data Entry Plugin for CREATE operation

```

```

dn: cn=PRE_DATA_ENTRY_MODIFY, cn=Plugins, cn=APPTYPE, cn=Applications,
   cn=Provisioning, cn=Directory Integration Platform, cn=Products,
   cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: Name_of_the_class_that_implements_the_plug-in
orclODIPPluginAddInfo: Pre Data Entry Plugin for MODIFY operation

```

```

dn: cn=POST_DATA_ENTRY_CREATE, cn=Plugins, cn=APPTYPE, cn=Applications,
   cn=Provisioning, cn=Directory Integration Platform, cn=Products,
   cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: Name_of_the_class_that_implements_the_plug-in
orclODIPPluginAddInfo: Post Data Entry Plugin for CREATE and modify operations

```

```

dn: cn=POST_DATA_ENTRY_MODIFY, cn=Plugins, cn=APPTYPE, cn=Applications,
   cn=Provisioning, cn=Directory Integration Platform, cn=Products,
   cn=OracleContext
changetype: add
objectClass: orclODIPPlugin

```

```

orclStatus: ENABLE
orclODIPPluginExecName: Name_of_the_class_that_implements_the_plug-in
orclODIPPluginAddInfo: Post Data Entry Plugin for MODIFY and CREATE operation

dn: cn=DATA_ACCESS, cn=Plugins, cn=APPTYPE, cn=Applications,
   cn=Provisioning, cn=Directory Integration Platform, cn=Products,
   cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: Name_of_the_class_that_implements_the_plug-in
orclODIPPluginAddInfo: Data Access Plugin

dn: cn=EVENT_DELIVERY_OUT, cn=Plugins, cn=APPTYPE, cn=Applications,
   cn=Provisioning, cn=Directory Integration Platform, cn=Products,
   cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: Name_of_the_class_that_implements_the_plug-in
orclODIPPluginAddInfo: Event Delivery Plugin for Outbound

dn: cn=EVENT_DELIVERY_IN, cn=Plugins, cn=APPTYPE, cn=Applications,
   cn=Provisioning, cn=Directory Integration Platform, cn=Products,
   cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: Name_of_the_class_that_implements_the_plug-in
orclODIPPluginAddInfo: Event Delivery Plugin for Inbound

```

Sample Code

```

/* Copyright (c) 2004, Oracle. All rights reserved. */
/**
DESCRIPTION
Sample PRE DATA Entry Plugin for CREATE operation that
validates the attribute.
PRIVATE CLASSES
None.
NOTES
This class implements the PRE_DATA_ENTRY_CREATE plugin ONLY
MODIFIED (MM/DD/YY)
12/15/04 \226 Creation
*/
package oracle.ldap.idm;

import java.util.*;
import javax.naming.*;
import javax.naming.ldap.*;
import javax.naming.directory.*;
import oracle.ldap.util.*;
import oracle.idm.provisioning.plugin.*;
/**
 * This class implements the PRE_DATA_ENTRY_CREATE plugin ONLY
 *
 */
public class SamplePreDataEntryCreatePlugin implements IDataEntryPlugin
{
    public ModPropertySet mpBaseUser = null;

```

```

public ModPropertySet mpAppUser = null;

public PluginStatus process(ApplicationContext appCtx, IdmUser idmuser,
    ModPropertySet baseUserAttr, ModPropertySet appUserAttr)
    throws PluginException
{
    PluginStatus retPluginStatus = null;
    String retProvStatus = null;
    String retProvStatusMsg = null;

    LDIFRecord lRec = null;
    LDIFAttribute lAttr = null;
    String val = null;
    if(null == baseUserAttr.getModPropertyValue("\223departmentNumber\224))
    {
        mpBaseUser = new ModPropertySet();
        mpBaseUser.addProperty("departmentNumber", "ST");
        appCtx.log("\223Base user attribute \226 departmentNumber missing\224 +
            \223Setting default - ST\224);
    }
    else if ( baseUserAttr.getModPropertyValue("\223departmentNumber\224)
        .notin("\223ST\224, \223APPS\224, \224CRM\224) )
    {
        throw new PluginException("\223Invalid department Number\224);
    }
    if((null == appUserAttr) ||
        null == appUserAttr.getModPropertyValue("\223emailQouta\224))
    {
        mpAppUser = new ModPropertySet();
        mpAppUser.addProperty("emailQouta", "50M");
        appCtx.log("\223Application user attribute - email Qouta missing \224 +
            \223Setting default - 50M\224);
    }
    return new PluginStatus(PluginStatus.SUCCESS, null, null);
}

public ModPropertySet getBaseAttrMods()
{
    return mpBaseUser;
}

public ModPropertySet getAppAttrMods()
{
    return mpAppUser;
}
}

/* Copyright (c) 2004, Oracle. All rights reserved. */
/**
DESCRIPTION
Sample POST DATA Entry Plugin for CREATE operation. Implementing a
policy check to provision only those users who belong to \223SALES\224.
PRIVATE CLASSES
None.
NOTES
This class implements the POST_DATA_ENTRY_CREATE plugin ONLY
MODIFIED (MM/DD/YY)
12/15/04 \226 Creation
*/
package oracle.ldap.idm;

```

```
import java.util.*;
import javax.naming.*;
import javax.naming.ldap.*;
import javax.naming.directory.*;
import oracle.ldap.util.*;
import oracle.idm.provisioning.plugin.*;
/**
 * This class implements the POST_DATA_ENTRY_CREATE plugin ONLY
 *
 */
public class SamplePostDataEntryCreatePlugin
{
    public ModPropertySet mpBaseUser = null;
    public ModPropertySet mpAppUser = null;

    public PluginStatus process(ApplicationContext appCtx, IdmUser idmuser,
        ModPropertySet baseUserAttr, ModPropertySet appUserAttr)
        throws PluginException
    {
        PluginStatus retPluginStatus = null;
        String retProvStatus = null;
        String retProvStatusMsg = null;

        if(null == baseUserAttr.getModPropertyValue("\223departmentNumber\224"))
        {
            mpBaseUser = new ModPropertySet();
            mpBaseUser.addProperty("departmentNumber ", "SALES");
            appCtx.log("Base user attribute \221c\222 is missing");

            retProvStatus = IdmUser.PROVISION_REQUIRED;
            retProvStatusMsg = "Provision policy: Only \221SALES\222\224."
        }
        else if (baseUserAttr.getModPropertyValue("\223departmentNumber\224")
            .equals("\223SALES\224"))
        {
            retProvStatus = IdmUser.PROVISION_REQUIRED;
            retProvStatusMsg = "Provision policy: Only \221SALES\222\224."
        }
        else
        {
            // do not provision those users who do not belong to SALES.
            retProvStatus = IdmUser.PROVISION_NOT_REQUIRED;
            retProvStatusMsg =
                "Do not provision the person who is not from \221SALES\222";
        }

        return new PluginStatus(PluginStatus.SUCCESS, retProvStatusMsg,
            retProvStatus);
    }

    public ModPropertySet getBaseAttrMods()
    {
        return mpBaseUser;
    }

    public ModPropertySet getAppAttrMods()
    {
        return mpAppUser;
    }
}
```

```
}

/* Copyright (c) 2004, Oracle. All rights reserved. */
/**
DESCRIPTION
Sample DATA Access Plugin.
NOTES
This class implements the DATA_ACCESS plugin
MODIFIED (MM/DD/YY)
12/15/04 \226 Creation
*/
package oracle.ldap.idm;

import javax.naming.*;
import javax.naming.ldap.*;
import javax.naming.directory.*;
import oracle.ldap.util.*;
import oracle.idm.provisioning.plugin.*;
/**
 * This class implements the DATA_ACCESS plugin ONLY
 *
 */
public class SampleDataAccessPlugin
{
    public PluginStatus process(ApplicationContext appCtx, IdmUser idmuser,
        ModPropertySet baseUserAttr, ModPropertySet appUserAttr)
        throws PluginException
    {
        try {
            DirContext dirCtx = appCtx.getDirCtx();
            if ( appCtx.getCallOp().equals(ApplicationContext.OP_CREATE )
                {
                    // Use the directory context and create the entry.
                }
            elseif ( appCtx.getCallOp().equals(ApplicationContext.OP_MODIFY)
                {
                    // Use the directory context and modify the entry.
                }
        } catch (Exception e) {
            throw new PluginException(e);
        }
        return new PluginStatus(PluginStatus.SUCCESS, null, null);
    }

    public PropertySet getAppUserData(ApplicationContext appCtx,
        IdmUser idmuser, String [] reqAttrs) throws PluginException
    {
        VarPropertySet vpSet = null;
        DirContext dirCtx = appCtx.getDirCtx();

        try {
            Attributes attrs= dirCtx.getAttributes("\223myAppContainer\224");
            vpSet = new VarPropertySet(); // Populate the VarPropertySet from attrs
        } catch(Exception ne) {
            throw new PluginException(e);
        }
        return vpSet;    }
}
```

This appendix contains the following sections:

- [Capabilities of DSML](#)
- [Benefits of DSML](#)
- [DSML Syntax](#)
- [Tools Enabled for DSML](#)

Capabilities of DSML

Directory services form a core part of distributed computing. XML is becoming the standard markup language for Internet applications. As directory services are brought to the Internet, there is a pressing and urgent need to express the directory information as XML data. This caters to the growing breed of applications that are not LDAP-aware yet require information exchange with a LDAP directory server.

Directory Services Mark-up Language (DSML) defines the XML representation of LDAP information and operations. The LDAP Data Interchange Format (LDIF) is used to convey directory information, or a set of changes to be applied to directory entries. The former is called Attribute Value Record and the latter is called Change Record.

Benefits of DSML

Using DSML with Oracle Internet Directory and Internet applications makes it easier to flexibly integrate data from disparate sources. Also, DSML enables applications that do not use LDAP to communicate with LDAP-based applications, easily operating on data generated by an Oracle Internet Directory client tool or accessing the directory through a firewall.

DSML is based on XML, which is optimized for delivery over the Web. Structured data in XML will be uniform and independent of application or vendors, thus making possible numerous new flat file type synchronization connectors. Once in XML format, the directory data can be made available in the middle tier and have more meaningful searches performed on it.

DSML Syntax

A DSML version 1 document describes either directory entries, a directory schema or both. Each directory entry has a unique name called a distinguished name (DN). A directory entry has a number of property-value pairs called directory attributes. Every directory entry is a member of a number of object classes. An entry's object classes

constrain the directory attributes the entry can take. Such constraints are described in a directory schema, which may be included in the same DSML document or may be in a separate document.

The following subsections briefly explain the top-level structure of DSML and how to represent the directory and schema entries.

Top-Level Structure

The top-level document element of DSML is of the type `dsml`, which may have child elements of the following types:

```
directory-entries
directory-schema
```

The child element `directory-entries` may in turn have child elements of the type `entry`. Similarly the child element `directory-schema` may in turn have child elements of the types `class` and `attribute-type`.

At the top level, the structure of a DSML document looks like this:

```
<!-- a document with directory & schema entries -->
<dsml:directory-entries>
  <dsml:entry dn="...">...</dsml:entry>
  .
  .
  .
</dsml:directory-entries>
.
.
.
<dsml:directory-schema>
  <dsml:class id="..." ...>...</dsml:class>
  <dsml:attribute-type id="..." ...>...</dsml:attribute-type>
  .
  .
  .
</dsml:directory-schema>
</dsml:dsml>
```

Directory Entries

The element type `entry` represents a directory entry in a DSML document. The `entry` element contains elements representing the entry's directory attributes. The distinguished name of the entry is indicated by the XML attribute `dn`.

Here is an XML entry to describe the directory entry:

```
<dsml:entry dn="uid=Heman, c=in, dc=oracle, dc=com">
<dsml:objectclass>
  <dsml:oc-value>top</dsml:oc-value>
  <dsml:oc-value ref="#person">person</dsml:oc-value>
  <dsml:oc-value>organizationalPerson</dsml:oc-value>
  <dsml:oc-value>inetOrgPerson</dsml:oc-value>
</dsml:objectclass>
<dsml:attr name="sn">
<dsml:value>Siva</dsml:value></dsml:attr>
<dsml:attr name="uid">
<dsml:value>Heman</dsml:value></dsml:attr>
<dsml:attr name="mail">
```



```

<dsml:attr name="givenname">
<dsml:value>Siva V. Kumar</dsml:value></dsml:attr>
<dsml:attr name="cn">
<dsml:value>SVK@oracle.com</dsml:value></dsml:attr>
<dsml:value>Siva Kumar</dsml:value></dsml:attr>

```

The `oc-value`'s `ref` is a URI Reference to a class element that defines the object class. In this case it is a URI [9] Reference to the element that defines the `person` object class. The child elements `objectclass` and `attr` are used to specify the object classes and the attributes of a directory entry.

Schema Entries

The element type `class` represents a schema entry in a DSML document. The `class` element takes an XML attribute `id` to make referencing easier.

For example, the object class definition for the `person` object class might look like the following:

```

<dsml:class id="person" superior="#top" type="structural">
  <dsml:name>person</dsml:name>
  <dsml:description>...</dsml:description>
  <dsml:object-identifier>2.5.6.6</object-identifier>
  <dsml:attribute ref="#sn" required="true"/>
  <dsml:attribute ref="#cn" required="true"/>
  <dsml:attribute ref="#userPassword" required="false"/>
  <dsml:attribute ref="#telephoneNumber" required="false"/>
  <dsml:attribute ref="#seeAlso" required="false"/>
  <dsml:attribute ref="#description" required="false"/>
</dsml:class>

```

The directory attributes are described in a similar way. For example, the attribute definition for the `cn` attribute may look like this:

```

<dsml:attribute-type id="cn">
  <dsml:name>cn</dsml:name>
  <dsml:description>...</dsml:description>
  <dsml:object-identifier>2.5.4.3</object-identifier>
  <dsml:syntax>1.3.6.1.4.1.1466.115.121.1.44</dsml:syntax>
</dsml:attribute-type>

```

Tools Enabled for DSML

With the XML framework, you can now use non-ldap applications to access directory data. The XML framework broadly defines the access points and provides the following tools:

- `ldapadd`
- `ldapaddmt`
- `ldapsearch`

See Also: "Oracle Internet Directory Server Administration Tools" in *Oracle Identity Management User Reference* for information about syntax and usage.

The client tool `ldifwrite` generates directory data and schema LDIF files. If you convert these LDIF files to XML, you can store the XML file on an application server

and query it. The query and response time is small compared to performing an LDAP operation against an LDAP server.

Glossary

3DES

See [Triple Data Encryption Standard \(3DES\)](#).

access control item (ACI)

Access control information represents the permissions that various entities or subjects have to perform operations on a given object in the directory. This information is stored in Oracle Internet Directory as user-modifiable operational [attributes](#), each of which is called an access control item (ACI). An ACI determines user access rights to directory data. It contains a set of rules for controlling access to entries (structural access items) and attributes (content access items). Access to both structural and content access items may be granted to one or more users or groups.

access control list (ACL)

A list of resources and the usernames of people who are permitted access to those resources within a computer system. In Oracle Internet Directory, an ACL is a list of [access control item \(ACI\) attribute values](#) that is associated with directory objects. The attribute values on that list represent the permissions that various directory user entities (or subjects) have on a given object.

access control policy point (ACP)

A directory entry that contains access control policy information that applies downward to all entries at lower positions in the [directory information tree \(DIT\)](#). This information affects the entry itself and all entries below it. In Oracle Internet Directory, you can create ACPs to apply an access control policy throughout a [subtree](#) of your directory.

account lockout

A security feature that locks a user account if repeated failed logon attempts occur within a specified amount of time, based on security policy settings. Account lockout occurs in OracleAS Single Sign-On when a user submits an account and password combination from any number of workstations more times than is permitted by Oracle Internet Directory. The default lockout period is 24 hours.

ACI

See [access control item \(ACI\)](#).

ACL

See [access control list \(ACL\)](#).

ACP

See [access control policy point \(ACP\)](#).

administrative area

A [subtree](#) on a directory server whose entries are under the control of a single administrative authority. The designated administrator controls each [entry](#) in that administrative area, as well as the directory [schema](#), [access control list \(ACL\)](#), and [attributes](#) for those entries.

Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES) is a [symmetric cryptography](#) algorithm that is intended to replace [Data Encryption Standard \(DES\)](#). AES is a Federal Information Processing Standard (FIPS) for the encryption of commercial and government data.

advanced replication

See [Oracle Database Advanced Replication](#).

advanced symmetric replication (ASR)

See [Oracle Database Advanced Replication](#).

AES

See [Advanced Encryption Standard \(AES\)](#).

anonymous authentication

The process by which a directory authenticates a user without requiring a user name and password combination. Each anonymous user then exercises the privileges specified for anonymous users.

API

See [application programming interface \(API\)](#).

application programming interface (API)

A series of software routines and development tools that comprise an interface between a computer application and lower-level services and functions (such as the operating system, device drivers, and other software applications). APIs serve as building blocks for programmers putting together software applications. For example, LDAP-enabled clients access Oracle Internet Directory information through programmatic calls available in the LDAP API.

application service provider

Application Service Providers (ASPs) are third-party entities that manage and distribute software-based services and solutions to customers across a wide area network from a central data center. In essence, ASPs are a way for companies to outsource some or almost all aspects of their information technology needs.

ASN.1

Abstract Syntax Notation One (ASN.1) is an International Telecommunication Union (ITU) notation used to define the syntax of information data. ASN.1 is used to describe structured information, typically information that is to be conveyed across some communications medium. It is widely used in the specification of Internet protocols.

ASR

See [Oracle Database Advanced Replication](#).

asymmetric algorithm

A **cryptographic algorithm** that uses different **keys** for **encryption** and **decryption**.

See also: **public key cryptography**.

asymmetric cryptography

See **public key cryptography**.

attribute

Directory attributes hold a specific data element such as a name, phone number, or job title. Each directory **entry** is comprised of a set of attributes, each of which belongs to an **object class**. Moreover, each attribute has both a *type*, which describes the kind of information in the attribute, and a *value*, which contains the actual data.

attribute configuration file

In an Oracle Directory Integration and Provisioning environment, a file that specifies attributes of interest in a connected directory.

attribute type

Attribute types specify information about a data element, such as the data type, maximum length, and whether it is single-valued or multivalued. The attribute type provides the real-world meaning for a value, and specifies the rules for creating and storing specific pieces of data, such as a name or an e-mail address.

attribute uniqueness

An Oracle Internet Directory feature that ensures that no two specified **attributes** have the same value. It enables applications synchronizing with the enterprise directory to use attributes as unique keys.

attribute value

Attribute values are the actual data contained within an **attribute** for a particular **entry**. For example, for the attribute type `email`, an attribute value might be `sally.jones@oracle.com`.

authentication

The process of verifying the identity claimed by an entity based on its credentials. Authentication of a user is generally based on something the user knows or has (for example, a password or a certificate).

Authentication of an electronic message involves the use of some kind of system (such as **public key cryptography**) to ensure that a file or message which claims to originate from a given individual or company actually does, and a check based on the contents of a message to ensure that it was not modified in transit.

authentication level

An OracleAS Single Sign-On parameter that enables you to specify a particular authentication behavior for an application. You can link this parameter with a specific **authentication plugin**.

authentication plugin

An implementation of a specific authentication method. OracleAS Single Sign-On has Java plugins for password authentication, digital certificates, Windows native authentication, and third-party access management.

authorization

The process of granting or denying access to a service or network resource. Most security systems are based on a two step process. The first stage is authentication, in which a user proves his or her identity. The second stage is authorization, in which a user is allowed to access various resources based on his or her identity and the defined [authorization policy](#).

authorization policy

Authorization policy describes how access to a protected resource is governed. Policy maps identities and objects to collections of rights according to some system model. For example, a particular authorization policy might state that users can access a sales report only if they belong to the sales group.

basic authentication

An [authentication](#) protocol supported by most browsers in which a Web server authenticates an entity with an encoded user name and password passed via data transmissions. Basic authentication is sometimes called plaintext authentication because the base-64 encoding can be decoded by anyone with a freely available decoding utility. Note that encoding is not the same as [encryption](#).

Basic Encoding Rules (BER)

Basic Encoding Rules (BER) are the standard rules for encoding data units set forth in [ASN.1](#). BER is sometimes incorrectly paired with ASN.1, which applies only to the abstract syntax description language, not the encoding technique.

BER

See [Basic Encoding Rules \(BER\)](#).

binding

In networking, binding is the establishment of a logical connection between communicating entities.

In the case of Oracle Internet Directory, binding refers to the process of authenticating to the directory.

The formal set of rules for carrying a [SOAP](#) message within or on top of another protocol (underlying protocol) for the purpose of exchange is also called a binding.

block cipher

Block ciphers are a type of [symmetric algorithm](#). A block cipher encrypts a message by breaking it down into fixed-size blocks (often 64 bits) and encrypting each block with a key. Some well known block ciphers include [Blowfish](#), [DES](#), and [AES](#).

See also: [stream cipher](#).

Blowfish

Blowfish is a [symmetric cryptography](#) algorithm developed by Bruce Schneier in 1993 as a faster replacement for [DES](#). It is a [block cipher](#) using 64-bit blocks and keys of up to 448 bits.

CA

See [Certificate Authority \(CA\)](#).

CA certificate

A **Certificate Authority (CA)** signs all certificates that it issues with its **private key**. The corresponding Certificate Authority's **public key** is itself contained within a certificate, called a CA Certificate (also referred to as a root certificate). A browser must contain the CA Certificate in its list of trusted root certificates in order to trust messages signed by the CA's private key.

cache

Generally refers to an amount of quickly accessible memory in your computer. However, on the Web it more commonly refers to where the browser stores downloaded files and graphics on the user's computer.

CBC

See **cipher block chaining (CBC)**.

central directory

In an Oracle Directory Integration and Provisioning environment, the directory that acts as the central repository. In an Oracle Directory Integration and Provisioning environment, Oracle Internet Directory is the central directory.

certificate

A certificate is a specially formatted data structure that associates a **public key** with the identity of its owner. A certificate is issued by a **Certificate Authority (CA)**. It contains the name, serial number, expiration dates, and public key of a particular entity. The certificate is digitally signed by the issuing CA so that a recipient can verify that the certificate is real. Most digital certificates conform to the **X.509** standard.

Certificate Authority (CA)

A Certificate Authority (CA) is a trusted third party that issues, renews, and revokes digital **certificates**. The CA essentially vouches for an entity's identity, and may delegate the verification of an applicant to a **Registration Authority (RA)**. Some well known Certificate Authorities (CAs) include Digital Signature Trust, Thawte, and VeriSign.

certificate chain

An ordered list of certificates containing one or more pairs of a user **certificate** and its associated **CA certificate**.

certificate management protocol (CMP)

Certificate Management Protocol (CMP) handles all relevant aspects of certificate creation and management. CMP supports interactions between **public key infrastructure (PKI)** components, such as the **Certificate Authority (CA)**, **Registration Authority (RA)**, and the user or application that is issued a certificate.

certificate request message format (CRMF)

Certificate Request Message Format (CRMF) is a format used for messages related to the life-cycle management of **X.509** certificates, as described in the **RFC 2511** specification.

certificate revocation list (CRL)

A Certificate Revocation List (CRL) is a list of digital **certificates** which have been revoked by the **Certificate Authority (CA)** that issued them.

change logs

A database that records changes made to a directory server.

cipher

See [cryptographic algorithm](#).

cipher block chaining (CBC)

Cipher block chaining (CBC) is a mode of operation for a [block cipher](#). CBC uses what is known as an initialization vector (IV) of a certain length. One of its key characteristics is that it uses a chaining mechanism that causes the decryption of a block of ciphertext to depend on all the preceding ciphertext blocks. As a result, the entire validity of all preceding blocks is contained in the immediately previous ciphertext block.

cipher suite

In [Secure Sockets Layer \(SSL\)](#), a set of authentication, encryption, and data integrity algorithms used for exchanging messages between network nodes. During an SSL handshake, the two nodes negotiate to see which cipher suite they will use when transmitting messages back and forth.

ciphertext

Ciphertext is the result of applying a [cryptographic algorithm](#) to readable data (plaintext) in order to render the data unreadable by all entities except those in possession of the appropriate [key](#).

circle of trust

A circle of trust is a [federation](#) of [service providers](#) and [identity providers](#) that have business relationships based on [Liberty Alliance](#) architecture and operational agreements, and with whom users can transact business in a secure and apparently seamless environment.

claim

A claim is a declaration made by an entity (for example, a name, identity, key, group, and so on).

client SSL certificates

A type of [certificate](#) used to identify a client machine to a server through [Secure Sockets Layer \(SSL\)](#) (client authentication).

cluster

A collection of interconnected usable whole computers that is used as a single computing resource. Hardware clusters provide high availability and scalability.

CMP

See [certificate management protocol \(CMP\)](#).

CMS

See [Cryptographic Message Syntax \(CMS\)](#).

code signing certificates

A type of [certificate](#) used to identify the entity who signed a Java program, Java Script, or other signed file.

cold backup

In Oracle Internet Directory, this refers to the procedure of adding a new **directory system agent (DSA)** node to an existing replicating system by using the database copy procedure.

concurrency

The ability to handle multiple requests simultaneously. Threads and processes are examples of concurrency mechanisms.

concurrent clients

The total number of clients that have established a session with Oracle Internet Directory.

concurrent operations

The number of operations that are being executed on Oracle Internet Directory from all of the **concurrent clients**. Note that this is not necessarily the same as the concurrent clients, because some of the clients may be keeping their sessions idle.

confidentiality

In cryptography, confidentiality (also known as privacy) is the ability to prevent unauthorized entities from reading data. This is typically achieved through **encryption**.

configset

See **configuration set entry**.

configuration set entry

An Oracle Internet Directory entry holding the configuration parameters for a specific instance of the directory server. Multiple configuration set entries can be stored and referenced at runtime. The configuration set entries are maintained in the subtree specified by the `subConfigsubEntry` attribute of the **directory-specific entry (DSE)**, which itself resides in the associated **directory information base (DIB)** against which the servers are started.

connect descriptor

A specially formatted description of the destination for a network connection. A connect descriptor contains destination service and network route information.

The destination service is indicated by using its service name for the Oracle Database or its Oracle System Identifier (SID) for Oracle release 8.0 or version 7 databases. The network route provides, at a minimum, the location of the listener through use of a network address.

connected directory

In an Oracle Directory Integration and Provisioning environment, an information repository requiring full synchronization of data between Oracle Internet Directory and itself—for example, an Oracle human resources database.

consumer

A directory server that is the destination of replication updates. Sometimes called a slave.

contention

Competition for resources.

context prefix

The **distinguished name (DN)** of the root of a **naming context**.

CRL

See **certificate revocation list (CRL)**.

CRMF

See **certificate request message format (CRMF)**.

cryptographic algorithm

A cryptographic algorithm is a defined sequence of processes to convert readable data (plaintext) to unreadable data (ciphertext) and vice versa. These conversions require some secret knowledge, normally contained in a **key**. Examples of cryptographic algorithms include **DES**, **AES**, **Blowfish**, and **RSA**.

Cryptographic Message Syntax (CMS)

Cryptographic Message Syntax (CMS) is a syntax defined in **RFC 3369** for signing, digesting, authenticating, and encrypting digital messages.

cryptography

The process of protecting information by transforming it into an unreadable format. The information is encrypted using a **key**, which makes the data unreadable, and is then decrypted later when the information needs to be used again. See also **public key cryptography** and **symmetric cryptography**.

dads.conf

A configuration file for Oracle HTTP Server that is used to configure a **database access descriptor (DAD)**.

DAS

See **Oracle Delegated Administration Services**. (DAS).

Data Encryption Standard (DES)

Data Encryption Standard (DES) is a widely used **symmetric cryptography** algorithm developed in 1974 by IBM. It applies a 56-bit key to each 64-bit block of data. DES and 3DES are typically used as encryption algorithms by **S/MIME**.

data integrity

The guarantee that the contents of the message received were not altered from the contents of the original message sent.

See also: **integrity**.

database access descriptor (DAD)

Database connection information for a particular Oracle Application Server component, such as the OracleAS Single Sign-On schema.

decryption

The process of converting the contents of an encrypted message (ciphertext) back into its original readable format (plaintext).

default identity management realm

In a hosted environment, one enterprise—for example, an application service provider—makes Oracle components available to multiple other enterprises and stores information for them. In such hosted environments, the enterprise performing the hosting is called the default identity management realm, and the enterprises that are hosted are each associated with their own identity management realm in the [directory information tree \(DIT\)](#).

default knowledge reference

A [knowledge reference](#) that is returned when the base object is not in the directory, and the operation is performed in a [naming context](#) not held locally by the server. A default knowledge reference typically sends the user to a server that has more knowledge about the directory partitioning arrangement.

default realm location

An attribute in the [root Oracle Context](#) that identifies the root of the [default identity management realm](#).

Delegated Administration Services

See [Oracle Delegated Administration Services](#).

delegated administrator

In a hosted environment, one enterprise—for example, an application service provider—makes Oracle components available to multiple other enterprises and stores information for them. In such an environment, a global administrator performs activities that span the entire directory. Other administrators—called delegated administrators—may exercise roles in specific identity management realms, or for specific applications.

DER

See [Distinguished Encoding Rules \(DER\)](#).

DES

See [Data Encryption Standard \(DES\)](#).

DIB

See [directory information base \(DIB\)](#).

Diffie-Hellman

Diffie-Hellman (DH) is a public key cryptography protocol that allows two parties to establish a shared secret over an unsecure communications channel. First published in 1976, it was the first workable public key cryptographic system.

See also: [symmetric algorithm](#).

digest

See [message digest](#).

digital certificate

See [certificate](#).

digital signature

A digital signature is the result of a two-step process applied to a given block of data. First, a [hash function](#) is applied to the data to obtain a result. Second, that result is

encrypted using the signer's **private key**. Digital signatures can be used to ensure integrity, message authentication, and non-repudiation of data. Examples of digital signature algorithms include **DSA**, **RSA**, and **ECDSA**.

Digital Signature Algorithm (DSA)

The Digital Signature Algorithm (DSA) is an **asymmetric algorithm** that is used as part of the Digital Signature Standard (DSS). It cannot be used for encryption, only for digital signatures. The algorithm produces a pair of large numbers that enable the authentication of the signatory, and consequently, the integrity of the data attached. DSA is used both in generating and verifying digital signatures.

See also: **Elliptic Curve Digital Signature Algorithm (ECDSA)**.

directory

See **Oracle Internet Directory**, **Lightweight Directory Access Protocol (LDAP)**, and **X.500**.

directory information base (DIB)

The complete set of all information held in the directory. The DIB consists of entries that are related to each other hierarchically in a **directory information tree (DIT)**.

directory information tree (DIT)

A hierarchical tree-like structure consisting of the **DNs** of the entries.

directory integration and provisioning server

In an Oracle Directory Integration and Provisioning environment, the server that drives the synchronization of data between Oracle Internet Directory and a **connected directory**.

directory integration profile

In an Oracle Directory Integration and Provisioning environment, an entry in Oracle Internet Directory that describes how Oracle Directory Integration and Provisioning communicates with external systems and what is communicated.

Directory Manager

See **Oracle Directory Manager**.

directory naming context

See **naming context**.

directory provisioning profile

A special kind of **directory integration profile** that describes the nature of provisioning-related notifications that Oracle Directory Integration and Provisioning sends to the directory-enabled applications.

directory replication group (DRG)

The directory servers participating in a **replication agreement**.

directory server instance

A discrete invocation of a directory server. Different invocations of a directory server, each started with the same or different configuration set entries and startup flags, are said to be different directory server instances.

directory synchronization profile

A special kind of [directory integration profile](#) that describes how synchronization is carried out between Oracle Internet Directory and an external system.

directory system agent (DSA)

The [X.500](#) term for a directory server.

directory-specific entry (DSE)

An entry specific to a directory server. Different directory servers may hold the same [directory information tree \(DIT\)](#) name, but have different contents—that is, the contents can be specific to the directory holding it. A DSE is an entry with contents specific to the directory server holding it.

directory user agent (DUA)

The software that accesses a directory service on behalf of the directory user. The directory user may be a person or another software element.

DIS

See [directory integration and provisioning server](#).

Distinguished Encoding Rules (DER)

Distinguished Encoding Rules (DER) are a set of rules for encoding [ASN.1](#) objects in byte-sequences. DER is a special case of [Basic Encoding Rules \(BER\)](#).

distinguished name (DN)

A [X.500](#) distinguished name (DN) is a unique name for a node in a directory tree. A DN is used to provide a unique name for a person or any other directory entry. A DN is a concatenation of selected [attributes](#) from each node in the tree along the path from the root node to the named entry's node. For example, in LDAP notation, the DN for a person named John Smith working at Oracle's US office would be: "cn=John Smith, ou=People, o=Oracle, c=us".

DIT

See [directory information tree \(DIT\)](#).

DN

See [distinguished name \(DN\)](#).

Document Type Definition (DTD)

A Document Type Definition (DTD) is a document that specifies constraints on the tags and tag sequences that are valid for a given [XML](#) document. DTDs follow the rules of Simple Generalized Markup Language (SGML), the parent language of XML.

domain component attribute

The domain component (dc) attribute can be used in constructing a [distinguished name \(DN\)](#) from a domain name. For example, using a domain name such as "oracle.com", one could construct a DN beginning with "dc=oracle, dc=com", and then use this DN as the root of its subtree of directory information.

DRG

See [directory replication group \(DRG\)](#).

DSA

See [Digital Signature Algorithm \(DSA\)](#) or [directory system agent \(DSA\)](#).

DSE

See [directory-specific entry \(DSE\)](#).

DTD

See [Document Type Definition \(DTD\)](#).

ECC

See [Elliptic Curve Cryptography \(ECC\)](#).

ECDSA

See [Elliptic Curve Digital Signature Algorithm \(ECDSA\)](#).

EJB

See [Enterprise Java Bean \(EJB\)](#).

Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography (ECC) is an alternative to the [RSA](#) encryption system which is based on the difficulty of solving elliptic curve discrete logarithm problems rather than on factoring large numbers. Developed and marketed by Certicom, ECC is especially suitable for environments, such as wireless devices and PC cards, where computational power is limited and high speed is required. For any given key size (measured in bits) ECC provides more security (is harder to decrypt without the key) than RSA.

Elliptic Curve Digital Signature Algorithm (ECDSA)

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analog of the [Digital Signature Algorithm \(DSA\)](#) standard. The advantages of ECDSA compared to RSA-like schemes are shorter key lengths and faster signing and decryption. For example, a 160 (210) bit ECC key is expected to give the same security as a 1024 (2048) bit RSA key, and the advantage increases as level of security is raised.

encryption

Encryption is the process of converting plaintext to ciphertext by applying a [cryptographic algorithm](#).

encryption certificate

An encryption certificate is a [certificate](#) containing a [public key](#) that is used to encrypt electronic messages, files, documents, or data transmission, or to establish or exchange a session key for these same purposes.

end-to-end security

This is a property of message-level security that is established when a message traverses multiple applications within and between business entities and is secure over its full route through and between the business entities.

Enterprise Java Bean (EJB)

Enterprise JavaBeans (EJBs) are a Java API developed by Sun Microsystems that defines a component architecture for multi-tier client/server systems. Because EJB systems are written in Java, they are platform independent. Being object oriented, they

can be implemented into existing systems with little or no recompiling and configuring.

Enterprise Manager

See [Oracle Enterprise Manager](#).

entry

An entry is a unique record in a directory that describes an object, such as a person. An entry consists of **attributes** and their associated **attribute values**, as dictated by the **object class** that describes that entry object. All entries in an LDAP directory structure are uniquely identified through their **distinguished name (DN)**.

export agent

In an Oracle Directory Integration and Provisioning environment, an agent that exports data out of Oracle Internet Directory.

export data file

In an Oracle Directory Integration and Provisioning environment, the file that contains data exported by an **export agent**.

export file

See [export data file](#).

external agent

A directory integration agent that is independent of Oracle Directory Integration and Provisioning server. Oracle Directory Integration and Provisioning server does not provide scheduling, mapping, or error handling services for it. An external agent is typically used when a third party metadirectory solution is integrated with Oracle Directory Integration and Provisioning.

external application

Applications that do not delegate authentication to the OracleAS Single Sign-On server. Instead, they display HTML login forms that ask for application user names and passwords. At the first login, users can choose to have the OracleAS Single Sign-On server retrieve these credentials for them. Thereafter, they are logged in to these applications transparently.

failover

The process of failure recognition and recovery. In an Oracle Application Server Cold Failover Cluster (Identity Management), an application running on one cluster node is transparently migrated to another cluster node. During this migration, clients accessing the service on the cluster see a momentary outage and may need to reconnect once the failover is complete.

fan-out replication

Also called a point-to-point replication, a type of replication in which a supplier replicates directly to a consumer. That consumer can then replicate to one or more other consumers. The replication can be either full or partial.

Federal Information Processing Standards (FIPS)

Federal Information Processing Standards (FIPS) are standards for information processing issued by the US government Department of Commerce's National Institute of Standards and Technology (NIST).

federated identity management (FIM)

The agreements, standards, and technologies that make identity and entitlements portable across autonomous domains. FIM makes it possible for an authenticated user to be recognized and take part in personalized services across multiple domains. It avoids pitfalls of centralized storage of personal information, while allowing users to link identity information between different accounts. Federated identity requires two key components: trust and standards. The trust model of federated identity management is based on [circle of trust](#). The standards are defined by the [Liberty Alliance](#) Project.

federation

A federation is a group of entities (companies and organizations) that have a shared user base, and have agreed to provide identity and authorization tokens so that their users only have to logon once to access all of the services in their [circle of trust](#). Within the federation, at least one entity serves as the [identity provider](#) who is responsible for authenticating users. Entities that provide services to the user are referred to as [service providers](#).

filter

A filter is an expression that defines the entries to be returned from a request or search on a directory. Filters are typically expressed as DNs, for example: `cn=susie smith,o=acme,c=us`.

FIM

See [federated identity management \(FIM\)](#).

FIPS

See [Federal Information Processing Standards \(FIPS\)](#).

forced authentication

The act of forcing a user to reauthenticate if he or she has been idle for a preconfigured amount of time. Oracle Application Server Single Sign-On enables you to specify a global user inactivity timeout. This feature is intended for installations that have sensitive applications.

GET

An authentication method whereby login credentials are submitted as part of the login URL.

global administrator

In a hosted environment, one enterprise—for example, an application service provider—makes Oracle components available to multiple other enterprises and stores information for them. In such an environment, a global administrator performs activities that span the entire directory.

global unique identifier (GUID)

An identifier generated by the system and inserted into an entry when the entry is added to the directory. In a multimaster replicated environment, the GUID, not the DN, uniquely identifies an entry. The GUID of an entry cannot be modified by a user.

global user inactivity timeout

An optional feature of Oracle Application Server Single Sign-On that forces users to reauthenticate if they have been idle for a preconfigured amount of time. The global user inactivity timeout is much shorter than the single sign-out session timeout.

globalization support

Multilanguage support for graphical user interfaces. Oracle Application Server Single Sign-On supports 29 languages.

globally unique user ID

A numeric string that uniquely identifies a user. A person may change or add user names, passwords, and distinguished names, but her globally unique user ID always remains the same.

grace login

A login occurring within the specified period before password expiration.

group search base

In the Oracle Internet Directory default [directory information tree \(DIT\)](#), the node in the identity management realm under which all the groups can be found.

guest user

One who is not an anonymous user, and, at the same time, does not have a specific user entry.

GUID

See [global unique identifier \(GUID\)](#).

handshake

A protocol two computers use to initiate a communication session.

hash

A number generated from a string of text with an algorithm. The hash value is substantially smaller than the text itself. Hash numbers are used for security and for faster access to data.

See also: [hash function](#).

hash function

In cryptography, a hash function or one-way hash function is an algorithm that produces a given value when applied to a given block of data. The result of a hash function can be used to ensure the integrity of a given block of data. For a hash function to be considered secure, it must be very difficult, given a known data block and a known result, to produce another data block that produces the same result.

Hashed Message Authentication Code (HMAC)

Hashed Message Authentication Code (HMAC) is a hash function technique used to create a secret hash function output. This strengthens existing hash functions such as MD5 and SHA. It is used in transport layer security (TLS).

HMAC

See [Hashed Message Authentication Code \(HMAC\)](#).

HTTP

The Hyper Text Transfer Protocol (HTTP) is the protocol used between a Web browser and a server to request a document and transfer its contents. The specification is maintained and developed by the World Wide Web Consortium.

HTTP Server

See [Oracle HTTP Server](#).

httpd.conf

The file used to configure [Oracle HTTP Server](#).

iASAdmins

The administrative group responsible for user and group management functions in Oracle Application Server. The OracleAS Single Sign-On administrator is a member of the group iASAdmins.

identity management

The process by which the complete security lifecycle for network entities is managed in an organization. It typically refers to the management of an organization's application users, where steps in the security life cycle include account creation, suspension, privilege modification, and account deletion. The network entities managed may also include devices, processes, applications, or anything else that needs to interact in a networked environment. Entities managed by an identity management process may also include users outside of the organization, for example customers, trading partners, or Web services.

identity management infrastructure database

The database that contains data for OracleAS Single Sign-On and Oracle Internet Directory.

identity management realm

A collection of identities, all of which are governed by the same administrative policies. In an enterprise, all employees having access to the intranet may belong to one realm, while all external users who access the public applications of the enterprise may belong to another realm. An identity management realm is represented in the directory by a specific [entry](#) with a special [object class](#) associated with it.

identity management realm-specific Oracle Context

An Oracle Context contained in each identity management realm. It stores the following information:

- User naming policy of the identity management realm—that is, how users are named and located.
- Mandatory authentication attributes.
- Location of groups in the identity management realm.
- Privilege assignments for the identity management realm—for example: who has privileges to add more users to the realm.
- Application specific data for that realm including authorizations.

identity provider

These are organizations recognized by the members of a [circle of trust](#) as the entity responsible for authenticating users and providing the digital identity information of

users to other parties in a **federation**. Identity providers enter into partnerships with service providers and provide services that follow agreed-upon practices set by all parties in a federation.

import agent

In an Oracle Directory Integration and Provisioning environment, an agent that imports data into Oracle Internet Directory.

import data file

In an Oracle Directory Integration and Provisioning environment, the file containing the data imported by an **import agent**.

infrastructure tier

The Oracle Application Server components responsible for identity management. These components are OracleAS Single Sign-On, Oracle Delegated Administration Services, and Oracle Internet Directory.

inherit

When an **object class** has been derived from another class, it also derives, or inherits, many of the characteristics of that other class. Similarly, an attribute subtype inherits the characteristics of its supertype.

instance

See **directory server instance**.

integrity

In cryptography, integrity is the ability to detect if data has been modified by entities that are not authorized to modify it.

Internet Directory

See **Oracle Internet Directory**.

Internet Engineering Task Force (IETF)

The principal body engaged in the development of new Internet standard specifications. It is an international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet.

Internet Message Access Protocol (IMAP)

A protocol allowing a client to access and manipulate electronic mail messages on a server. It permits manipulation of remote message folders, also called mailboxes, in a way that is functionally equivalent to local mailboxes.

J2EE

See **Java 2 Platform, Enterprise Edition (J2EE)**.

Java 2 Platform, Enterprise Edition (J2EE)

Java 2 Platform, Enterprise Edition (J2EE) is an environment for developing and deploying enterprise applications, defined by Sun Microsystems Inc. The J2EE platform consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing multitiered, Web-based applications.

Java Server Page (JSP)

JavaServer Pages (JSP), a server-side technology, are an extension to the Java servlet technology that was developed by Sun Microsystems. JSPs have dynamic scripting capability that works in tandem with HTML code, separating the page logic from the static elements (the design and display of the page). Embedded in the HTML page, the Java source code and its extensions help make the HTML more functional, being used in dynamic database queries, for example.

JSP

See [Java Server Page \(JSP\)](#).

key

A key is a data structure that contains some secret knowledge necessary to successfully encrypt or decrypt a given block of data. The larger the key, the harder it is to crack a block of encrypted data. For example, a 256-bit key is more secure than a 128-bit key.

key pair

A [public key](#) and its associated [private key](#).

See also: [public/private key pair](#).

knowledge reference

The access information (name and address) for a remote [directory system agent \(DSA\)](#) and the name of the [directory information tree \(DIT\)](#) subtree that the remote DSA holds. Knowledge references are also called referrals.

latency

The time a client has to wait for a given directory operation to complete. Latency can be defined as wasted time. In networking discussions, latency is defined as the travel time of a packet from source to destination.

LDAP

See [Lightweight Directory Access Protocol \(LDAP\)](#).

LDAP connection cache

To improve throughput, the OracleAS Single Sign-On server caches and then reuses connections to Oracle Internet Directory.

LDAP Data Interchange Format (LDIF)

A common, text-based format for exchanging directory data between systems. The set of standards for formatting an input file for any of the LDAP command-line utilities.

LDIF

See [LDAP Data Interchange Format \(LDIF\)](#).

legacy application

Older application that cannot be modified to delegate authentication to the OracleAS Single Sign-On server. Also known as an [external application](#).

Liberty Alliance

The Liberty Alliance Project is an alliance of more than 150 companies, non-profit, and government organizations from around the globe. The consortium is committed to developing an open standard for federated network identity that supports all current

and emerging network devices. The Liberty Alliance is the only global body working to define and drive open technology standards, privacy, and business guidelines for [federated identity management \(FIM\)](#).

Lightweight Directory Access Protocol (LDAP)

A set of protocols for accessing information in directories. LDAP supports TCP/IP, which is necessary for any type of Internet access. Its framework of design conventions supports industry-standard directory products, such as Oracle Internet Directory. Because it is a simpler version of the [X.500](#) standard, LDAP is sometimes called X.500 light.

load balancer

Hardware devices and software that balance connection requests between two or more servers, either due to heavy load or failover. BigIP, Alteon, or Local Director are all popular hardware devices. Oracle Application Server Web Cache is an example of load balancing software.

logical host

In an Oracle Application Server Cold Failover Cluster (Identity Management), one or more disk groups and pairs of host names and IP addresses. It is mapped to a physical host in the cluster. This physical host impersonates the host name and IP address of the logical host.

MAC

See [message authentication code \(MAC\)](#).

man-in-the-middle

A security attack characterized by the third-party, surreptitious interception of a message. The third-party, the *man-in-the-middle*, decrypts the message, re-encrypts it (with or without alteration of the original message), and retransmits it to the originally-intended recipient—all without the knowledge of the legitimate sender and receiver. This type of security attack works only in the absence of [authentication](#).

mapping rules file

In an Oracle Directory Integration and Provisioning environment, the file that specifies mappings between Oracle Internet Directory attributes and those in a [connected directory](#).

master definition site (MDS)

In replication, a master definition site is the Oracle Internet Directory database from which the administrator runs the configuration scripts.

master site

In replication, a master site is any site other than the [master definition site \(MDS\)](#) that participates in LDAP replication.

matching rule

In a search or compare operation, determines equality between the attribute value sought and the attribute value stored. For example, matching rules associated with the `telephoneNumber` attribute could cause "(650) 123-4567" to be matched with either "(650) 123-4567" or "6501234567" or both. When you create an [attribute](#), you associate a matching rule with it.

MD2

Message Digest Two (MD2) is a message digest [hash function](#). The algorithm processes input text and creates a 128-bit [message digest](#) which is unique to the message and can be used to verify data integrity. MD2 was developed by Ron Rivest for RSA Security and is intended to be used in systems with limited memory, such as smart cards.

MD4

Message Digest Four (MD4) is similar to [MD2](#) but designed specifically for fast processing in software.

MD5

Message Digest Five (MD5) is a message digest [hash function](#). The algorithm processes input text and creates a 128-bit [message digest](#) which is unique to the message and can be used to verify data integrity. MD5 was developed by Ron Rivest after potential weaknesses were reported in [MD4](#). MD5 is similar to MD4 but slower because more manipulation is made to the original data.

MDS

See [master definition site \(MDS\)](#).

message authentication

The process of verifying that a particular message came from a particular entity.

See also: [authentication](#).

message authentication code (MAC)

The Message Authentication Code (MAC) is a result of a two-step process applied to a given block of data. First, the result of a [hash function](#) is obtained. Second, that result is encrypted using a [secret key](#). The MAC can be used to authenticate the source of a given block of data.

message digest

The result of a [hash function](#).

See also: [hash](#).

metadirectory

A directory solution that shares information between all enterprise directories, integrating them into one virtual directory. It centralizes administration, thereby reducing administrative costs. It synchronizes data between directories, thereby ensuring that it is consistent and up-to-date across the enterprise.

middle tier

That portion of a OracleAS Single Sign-On instance that consists of the Oracle HTTP Server and OC4J. The OracleAS Single Sign-On middle tier is situated between the identity management infrastructure database and the client.

mod_osso

A module on the Oracle HTTP Server that enables applications protected by OracleAS Single Sign-On to accept HTTP headers in lieu of a user name and password once the user has logged into the OracleAS Single Sign-On server. The values for these headers are stored in the [mod_osso cookie](#).

mod_osso cookie

User data stored on the HTTP server. The cookie is created when a user authenticates. When the same user requests another application, the Web server uses the information in the mod_osso cookie to log the user in to the application. This feature speeds server response time.

mod_proxy

A module on the Oracle HTTP Server that makes it possible to use [mod_osso](#) to enable single sign-on to legacy, or [external applications](#).

MTS

See [shared server](#).

multimaster replication

Also called peer-to-peer or *n*-way replication, a type of replication that enables multiple sites, acting as equals, to manage groups of replicated data. In a multimaster replication environment, each node is both a supplier and a consumer node, and the entire directory is replicated on each node.

naming attribute

The attribute used to compose the RDN of a new user entry created through Oracle Delegated Administration Services or Oracle Internet Directory Java APIs. The default value for this is `cn`.

naming context

A subtree that resides entirely on one server. It must be contiguous, that is, it must begin at an entry that serves as the top of the subtree, and extend downward to either leaf entries or [knowledge references](#) (also called referrals) to subordinate naming contexts. It can range in size from a single entry to the entire [directory information tree \(DIT\)](#).

native agent

In an Oracle Directory Integration and Provisioning environment, an agent that runs under the control of the [directory integration and provisioning server](#). It is in contrast to an [external agent](#).

net service name

A simple name for a service that resolves to a connect descriptor. Users initiate a connect request by passing a user name and password along with a net service name in a connect string for the service to which they wish to connect, for example:

```
CONNECT username/password@net_service_name
```

Depending on your needs, net service names can be stored in a variety of places, including:

- Local configuration file, `tnsnames.ora`, on each client
- Directory server
- Oracle Names server
- External naming service, such as NDS, NIS or CDS

Net Services

See [Oracle Net Services](#).

nickname attribute

The attribute used to uniquely identify a user in the entire directory. The default value for this is `uid`. Applications use this to resolve a simple user name to the complete distinguished name. The user nickname attribute cannot be multi-valued—that is, a given user cannot have multiple nicknames stored under the same attribute name.

non-repudiation

In cryptography, the ability to prove that a given **digital signature** was produced with a given entity's **private key**, and that a message was sent untampered at a given point in time.

OASIS

Organization for the Advancement of Structured Information Standards. OASIS is a worldwide not-for-profit consortium that drives the development, convergence and adoption of e-business standards.

object class

In LDAP, object classes are used to group information. Typically an object class models a real-world object such as a person or a server. Each directory entry belongs to one or more object classes. The object class determines the attributes that make up an entry. One object class can be derived from another, thereby inheriting some of the characteristics of the other class.

OC4J

See [Oracle Containers for J2EE \(OC4J\)](#).

OCA

See [Oracle Certificate Authority](#).

OCI

See [Oracle Call Interface \(OCI\)](#).

OCSP

See [Online Certificate Status Protocol \(OCSP\)](#).

OEM

See [Oracle Enterprise Manager](#).

OID

See [Oracle Internet Directory](#).

OID Control Utility

A command-line tool for issuing run-server and stop-server commands. The commands are interpreted and executed by the **OID Monitor** process.

OID Database Password Utility

The utility used to change the password with which Oracle Internet Directory connects to an Oracle Database.

OID Monitor

The Oracle Internet Directory component that initiates, monitors, and terminates the Oracle Internet Directory Server processes. It also controls the replication server if one is installed, and Oracle Directory Integration and Provisioning Server.

Online Certificate Status Protocol (OCSP)

Online Certificate Status Protocol (OCSP) is one of two common schemes for checking the validity of digital certificates. The other, older method, which OCSP has superseded in some scenarios, is [certificate revocation list \(CRL\)](#). OCSP is specified in [RFC 2560](#).

one-way function

A function that is easy to compute in one direction but quite difficult to reverse compute, that is, to compute in the opposite direction.

one-way hash function

A [one-way function](#) that takes a variable sized input and creates a fixed size output.

See also: [hash function](#).

Oracle Application Server Single Sign-On

OracleAS Single Sign-On consists of program logic that enables you to log in securely to applications such as expense reports, mail, and benefits. These applications take two forms: [partner applications](#) and [external applications](#). In both cases, you gain access to several applications by authenticating only once.

Oracle Call Interface (OCI)

An application programming interface (API) that enables you to create applications that use the native procedures or function calls of a third-generation language to access an Oracle Database server and control all phases of SQL statement execution.

Oracle Certificate Authority

Oracle Application Server Certificate Authority is a [Certificate Authority \(CA\)](#) for use within your Oracle Application Server environment. OracleAS Certificate Authority uses Oracle Internet Directory as the storage repository for certificates. OracleAS Certificate Authority integration with OracleAS Single Sign-On and Oracle Internet Directory provides seamless certificate provisioning mechanisms for applications relying on them. A user provisioned in Oracle Internet Directory and authenticated in OracleAS Single Sign-On can choose to request a digital certificate from OracleAS Certificate Authority.

Oracle CMS

Oracle CMS implements the IETF [Cryptographic Message Syntax \(CMS\)](#) protocol. CMS defines data protection schemes that allow for secure message envelopes.

Oracle Containers for J2EE (OC4J)

A lightweight, scalable container for [Java 2 Platform, Enterprise Edition \(J2EE\)](#).

Oracle Context

See [identity management realm-specific Oracle Context](#) and [root Oracle Context](#).

Oracle Crypto

Oracle Crypto is a pure Java library that provides core cryptography algorithms.

Oracle Database Advanced Replication

A feature in the Oracle Database that enables database tables to be kept synchronized across two Oracle databases.

Oracle Delegated Administration Services

A set of individual, pre-defined services—called Oracle Delegated Administration Services units—for performing directory operations on behalf of a user. Oracle Internet Directory Self-Service Console makes it easier to develop and deploy administration solutions for both Oracle and third-party applications that use Oracle Internet Directory.

Oracle Directory Integration and Provisioning

A collection of interfaces and services for integrating multiple directories by using Oracle Internet Directory and several associated plug-ins and connectors. A feature of Oracle Internet Directory that enables an enterprise to use an external user repository to authenticate to Oracle products.

Oracle Directory Integration and Provisioning Server

In an Oracle Directory Integration and Provisioning environment, a daemon process that monitors Oracle Internet Directory for change events and takes action based on the information present in the [directory integration profile](#).

Oracle Directory Integration Platform

A component of [Oracle Internet Directory](#). It is a framework developed to integrate applications around a central LDAP directory like Oracle Internet Directory.

Oracle Directory Manager

A Java-based tool with a graphical user interface for administering Oracle Internet Directory.

Oracle Enterprise Manager

A separate Oracle product that combines a graphical console, agents, common services, and tools to provide an integrated and comprehensive systems management platform for managing Oracle products.

Oracle HTTP Server

Software that processes Web transactions that use the Hypertext Transfer Protocol (HTTP). Oracle uses HTTP software developed by the Apache Group.

Oracle Identity Management

An infrastructure enabling deployments to manage centrally and securely all enterprise identities and their access to various applications in the enterprise.

Oracle Internet Directory

A general purpose directory service that enables retrieval of information about dispersed users and network resources. It combines [Lightweight Directory Access Protocol \(LDAP\)](#) Version 3 with the high performance, scalability, robustness, and availability of the Oracle Database.

Oracle Liberty SDK

Oracle Liberty SDK implements the [Liberty Alliance](#) Project specifications enabling federated single sign-on between third-party Liberty-compliant applications.

Oracle Net Services

The foundation of the Oracle family of networking products, allowing services and their client applications to reside on different computers and communicate. The main function of Oracle Net Services is to establish network sessions and transfer data

between a client application and a server. Oracle Net Services is located on each computer in the network. Once a network session is established, Oracle Net Services acts as a data courier for the client and the server.

Oracle PKI certificate usages

Defines Oracle application types that a [certificate](#) supports.

Oracle PKI SDK

Oracle PKI SDK implements the security protocols that are necessary within [public key infrastructure \(PKI\)](#) implementations.

Oracle SAML

Oracle SAML provides a framework for the exchange of security credentials among disparate systems and applications in an XML-based format as outlined in the [OASIS](#) specification for the [Security Assertions Markup Language \(SAML\)](#).

Oracle Security Engine

Oracle Security Engine extends Oracle Crypto by offering X.509 based certificate management functions. Oracle Security Engine is a superset of Oracle Crypto.

Oracle S/MIME

Oracle S/MIME implements the [Secure/Multipurpose Internet Mail Extension \(S/MIME\)](#) specifications from the [Internet Engineering Task Force \(IETF\)](#) for secure e-mail.

Oracle Wallet Manager

A Java-based application that security administrators use to manage public-key security credentials on clients and servers.

See also: *Oracle Advanced Security Administrator's Guide*.

Oracle Web Services Security

Oracle Web Services Security provides a framework for authentication and authorization using existing security technologies as outlined in the [OASIS](#) specification for Web Services Security.

Oracle XML Security

Oracle XML Security implements the W3C specifications for XML Encryption and XML Signature.

OracleAS Portal

An OracleAS Single Sign-On [partner application](#) that provides a mechanism for integrating files, images, applications, and Web sites. The External Applications portlet provides access to external applications.

other information repository

In an Oracle Directory Integration and Provisioning environment, in which Oracle Internet Directory serves as the [central directory](#), any information repository except Oracle Internet Directory.

OWM

See [Oracle Wallet Manager](#).

partition

A unique, non-overlapping directory naming context that is stored on one directory server.

partner application

An Oracle Application Server application or non-Oracle application that delegates the authentication function to the OracleAS Single Sign-On server. This type of application spares users from reauthenticating by accepting **mod_osso** headers.

peer-to-peer replication

Also called multimaster replication or *n*-way replication. A type of replication that enables multiple sites, acting as equals, to manage groups of replicated data. In such a replication environment, each node is both a supplier and a consumer node, and the entire directory is replicated on each node.

PKCS#1

The Public Key Cryptography Standards (PKCS) are specifications produced by RSA Laboratories. PKCS#1 provides recommendations for the implementation of public-key cryptography based on the RSA algorithm, covering the following aspects: cryptographic primitives; encryption schemes; signature schemes; ASN.1 syntax for representing keys and for identifying the schemes.

PKCS#5

The Public Key Cryptography Standards (PKCS) are specifications produced by RSA Laboratories. PKCS#5 provides recommendations for the implementation of password-based cryptography.

PKCS#7

The Public Key Cryptography Standards (PKCS) are specifications produced by RSA Laboratories. PKCS #7 describes general syntax for data that may have cryptography applied to it, such as digital signatures and digital envelopes.

PKCS#8

The Public Key Cryptography Standards (PKCS) are specifications produced by RSA Laboratories. PKCS #8 describes syntax for private key information, including a private key for some public key algorithms and a set of attributes. The standard also describes syntax for encrypted private keys.

PKCS#10

The Public Key Cryptography Standards (PKCS) are specifications produced by RSA Laboratories. PKCS #10 describes syntax for a request for certification of a public key, a name, and possibly a set of attributes.

PKCS#12

The Public Key Cryptography Standards (PKCS) are specifications produced by RSA Laboratories. PKCS #12 describes a transfer syntax for personal identity information, including private keys, certificates, miscellaneous secrets, and extensions. Systems (such as browsers or operating systems) that support this standard allow a user to import, export, and exercise a single set of personal identity information—typically in a format called a **wallet**.

PKI

See **public key infrastructure (PKI)**.

plaintext

Plaintext is readable data prior to a transformation to ciphertext using encryption, or readable data that is the result of a transformation from ciphertext using decryption.

point-to-point replication

Also called fan-out replication is a type of replication in which a supplier replicates directly to a consumer. That consumer can then replicate to one or more other consumers. The replication can be either full or partial.

policy precedence

In Oracle Application Server Certificate Authority (OCA), policies are applied to incoming requests in the order that they are displayed on the main policy page. When the OCA policy processor module parses policies, those that appear toward the top of the policy list are applied to requests first. Those that appear toward the bottom of the list are applied last and take precedence over the others. Only enabled policies are applied to incoming requests.

policy.properties

A multipurpose configuration file for Oracle Application Server Single Sign-On that contains basic parameters required by the single sign-on server. Also used to configure advanced features of OracleAS Single Sign-On, such as multilevel authentication.

POSIX

Portable Operating System Interface for UNIX. A set of programming interface standards governing how to write application source code so that the applications are portable between operating systems. A series of standards being developed by the [Internet Engineering Task Force \(IETF\)](#).

POST

An authentication method whereby login credentials are submitted within the body of the login form.

predicates

In Oracle Application Server Certificate Authority (OCA), a policy predicate is a logical expression that can be applied to a policy to limit how it is applied to incoming certificate requests or revocations. For example, the following predicate expression specifies that the policy in which it appears can have a different effect for requests or revocations from clients with DNs that include "ou=sales,o=acme,c=us":

```
Type=="client" AND DN=="ou=sales,o=acme,c=us"
```

primary node

In an Oracle Application Server Cold Failover Cluster (Identity Management), the cluster node on which the application runs at any given time.

See also: [secondary node](#).

private key

A private key is the secret key in a [public/private key pair](#) used in [public key cryptography](#). An entity uses its private key to decrypt data that has been encrypted with its [public key](#). The entity can also use its private key to create [digital signatures](#). The security of data encrypted with the entity's public key as well as signatures created by the private key depends on the private key remaining secret.

private key cryptography

See [symmetric cryptography](#).

profile

See [directory integration profile](#).

provisioned applications

Applications in an environment where user and group information is centralized in Oracle Internet Directory. These applications are typically interested in changes to that information in Oracle Internet Directory.

provisioning

The process of providing users with access to applications and other resources that may be available in an enterprise environment.

provisioning agent

An application or process that translates Oracle-specific provisioning events to external or third-party application-specific events.

provisioning integration profile

A special kind of [directory integration profile](#) that describes the nature of provisioning-related notifications that Oracle Directory Integration and Provisioning sends to the directory-enabled applications.

proxy server

A server between a client application, such as a Web browser, and a real server. It intercepts all requests to the real server to see if it can fulfil the requests itself. If not, it forwards the request to the real server. In OracleAS Single Sign-On, proxies are used for load balancing and as an extra layer of security.

See also: [load balancer](#).

proxy user

A kind of user typically employed in an environment with a middle tier such as a firewall. In such an environment, the end user authenticates to the middle tier. The middle tier then logs into the directory on the end user's behalf. A proxy user has the privilege to switch identities and, once it has logged into the directory, switches to the end user's identity. It then performs operations on the end user's behalf, using the authorization appropriate to that particular end user.

public key

A public key is the non-secret key in a [public/private key pair](#) used in [public key cryptography](#). A public key allows entities to encrypt data that can only then be decrypted with the public key's owner using the corresponding [private key](#). A public key can also be used to verify digital signatures created with the corresponding private key.

public key certificate

See [certificate](#).

public key cryptography

Public key cryptography (also known as asymmetric cryptography) uses two keys, one public and the other private. These keys are called a key pair. The private key must be kept secret, while the public key can be transmitted to any party. The private key and

the public key are mathematically related. A message that is signed by a private key can be verified by the corresponding public key. Similarly, a message encrypted by the public key can be decrypted by the private key. This method ensures privacy because only the owner of the private key can decrypt the message.

public key encryption

The process in which the sender of a message encrypts the message with the public key of the recipient. Upon delivery, the message is decrypted by the recipient using the recipient's private key.

public key infrastructure (PKI)

A public key infrastructure (PKI) is a system that manages the issuing, distribution, and authentication of **public keys** and **private keys**. A PKI typically comprises the following components:

- A **Certificate Authority (CA)** that is responsible for generating, issuing, publishing and revoking digital certificates.
- A **Registration Authority (RA)** that is responsible for verifying the information supplied in requests for certificates made to the CA.
- A directory service where a **certificate** or **certificate revocation list (CRL)** gets published by the CA and where they can be retrieved by relying third parties.
- Relying third parties that use the certificates issued by the CA and the **public keys** contained therein to verify **digital signatures** and encrypt data.

public/private key pair

A mathematically related set of two numbers where one is called the private key and the other is called the public key. Public keys are typically made widely available, while private keys are available only to their owners. Data encrypted with a public key can only be decrypted with its associated private key and vice versa. Data encrypted with a public key cannot be decrypted with the same public key.

RC2

Rivest Cipher Two (RC2) is a 64-bit **block cipher** developed by Ronald Rivest for RSA Security, and was designed as a replacement for **Data Encryption Standard (DES)**.

RC4

Rivest Cipher Four (RC4) is a **stream cipher** developed by Ronald Rivest for RSA Security. RC4 allows variable key lengths up to 1024 bits. RC4 is most commonly used to secure data communications by encrypting traffic between Web sites that use the **Secure Sockets Layer (SSL)** protocol.

RDN

See **relative distinguished name (RDN)**.

readable data

Data prior to a transformation to ciphertext via encryption or data that is the result of a transformation from ciphertext via decryption.

realm

See **identity management realm**.

realm search base

An attribute in the **root Oracle Context** that identifies the entry in the **directory information tree (DIT)** that contains all **identity management realms**. This attribute is used when mapping a simple realm name to the corresponding entry in the directory.

referral

Information that a directory server provides to a client and which points to other servers the client must contact to find the information it is requesting.

See also: **knowledge reference**.

Registration Authority (RA)

The Registration Authority (RA) is responsible for verifying and enrolling users before a certificate is issued by a **Certificate Authority (CA)**. The RA may assign each applicant a relative distinguished value or name for the new certificate applied. The RA does not sign or issue certificates.

registry entry

An entry containing runtime information associated with invocations of Oracle Internet Directory servers, called a **directory server instance**. Registry entries are stored in the directory itself, and remain there until the corresponding directory server instance stops.

relational database

A structured collection of data that stores data in tables consisting of one or more rows, each containing the same set of columns. Oracle makes it very easy to link the data in multiple tables. This is what makes Oracle a relational database management system, or RDBMS. It stores data in two or more tables and enables you to define relationships between the tables. The link is based on one or more fields common to both tables.

relative distinguished name (RDN)

The local, most granular level entry name. It has no other qualifying entry names that would serve to uniquely address the entry. In the example, `cn=Smith,o=acme,c=US`, the RDN is `cn=Smith`.

remote master site (RMS)

In a replicated environment, any site, other than the **master definition site (MDS)**, that participates in **Oracle Database Advanced Replication**.

replica

Each copy of a **naming context** that is contained within a single server.

replication agreement

A special directory entry that represents the replication relationship among the directory servers in a **directory replication group (DRG)**.

response time

The time between the submission of a request and the completion of the response.

RFC

The Internet Request For Comments (or RFC) documents are the written definitions of the protocols and policies of the Internet. The Internet Engineering Task Force (IETF) facilitates the discussion, development, and establishment of new standards. A

standard is published using the RFC acronym and a reference number. For example, the official standard for e-mail is RFC 822.

root CA

In a hierarchical [public key infrastructure \(PKI\)](#), the root [Certificate Authority \(CA\)](#) is the CA whose [public key](#) serves as the most trusted datum for a security domain.

root directory specific entry (DSE)

An entry storing operational information about the directory. The information is stored in a number of attributes.

root DSE

See [root directory specific entry \(DSE\)](#).

root Oracle Context

In the Oracle Identity Management infrastructure, the root Oracle Context is an entry in Oracle Internet Directory containing a pointer to the default identity management realm in the infrastructure. It also contains information on how to locate an identity management realm given a simple name of the realm.

RSA

RSA is a [public key cryptography](#) algorithm named after its inventors (Rivest, Shamir, and Adelman). The RSA algorithm is the most commonly used encryption and authentication algorithm and is included as part of the Web browsers from Netscape and Microsoft, and many other products.

RSAES-OAEP

The RSA Encryption Scheme - Optimal Asymmetric Encryption Padding (RSAES-OAEP) is a public key encryption scheme combining the [RSA](#) algorithm with the OAEP method. Optimal Asymmetric Encryption Padding (OAEP) is a method for encoding messages developed by Mihir Bellare and Phil Rogaway.

S/MIME

See [Secure/Multipurpose Internet Mail Extension \(S/MIME\)](#).

SAML

See [Security Assertions Markup Language \(SAML\)](#).

SASL

See [Simple Authentication and Security Layer \(SASL\)](#).

scalability

The ability of a system to provide throughput in proportion to, and limited only by, available hardware resources.

schema

The collection of [attributes](#), [object classes](#), and their corresponding [matching rules](#).

secondary node

In an Oracle Application Server Cold Failover Cluster (Identity Management), the cluster node to which an application is moved during a failover.

See also: [primary node](#).

secret key

A secret key is the **key** used in a **symmetric algorithm**. Since a secret key is used for both encryption and decryption, it must be shared between parties that are transmitting ciphertext to one another but must be kept secret from all unauthorized entities.

secret key cryptography

See **symmetric cryptography**.

Secure Hash Algorithm (SHA)

Secure Hash Algorithm (SHA) is a **hash function** algorithm that produces a 160-bit **message digest** based upon the input. The algorithm is used in the Digital Signature Standard (DSS). With the introduction of the Advanced Encryption Standard (AES) which offers three key sizes: 128, 192 and 256 bits, there has been a need for a companion hash algorithm with a similar level of security. The newer SHA-256, SHA-284 and SHA-512 hash algorithms comply with these enhanced requirements.

Secure Sockets Layer (SSL)

Secure Sockets Layer (SSL) is a protocol designed by Netscape Communications to enable encrypted, authenticated communications across networks (such as the Internet). SSL uses the **public key encryption** system from RSA, which also includes the use of a digital certificate. SSL provides three elements of secure communications: **confidentiality**, **authentication**, and **integrity**.

SSL has evolved into **Transport Layer Security (TLS)**. TLS and SSL are not interoperable. However, a message sent with TLS can be handled by a client that handles SSL.

Secure/Multipurpose Internet Mail Extension (S/MIME)

Secure/Multipurpose Internet Mail Extension (S/MIME) is an Internet Engineering Task Force (IETF) standard for securing MIME data through the use of **digital signatures** and **encryption**.

Security Assertions Markup Language (SAML)

Security Assertions Markup Language (SAML) is an **XML**-based framework for exchanging security information over the Internet. SAML enables the exchange of **authentication** and **authorization** information between various security services systems that otherwise would not be able to interoperate. The SAML 1.0 specification was adopted by **OASIS** in 2002.

server certificate

A **certificate** that attests to the identity of an organization that uses a secure Web server to serve data. A server certificate must be associated with a **public/private key pair** issued by a mutually trusted **Certificate Authority (CA)**. Server certificates are required for secure communications between a browser and a Web server.

service provider

These are organizations recognized by the members of a **circle of trust** as the entities that provide Web-based services to users. Service providers enter into partnerships with other service providers and identity providers with the goal of providing their common users with secure single sign-on between all parties of the **federation**.

service time

The time between the initiation of a request and the completion of the response to the request.

session key

A **secret key** that is used for the duration of one message or communication session.

SGA

See **System Global Area (SGA)**.

SHA

See **Secure Hash Algorithm (SHA)**.

shared server

A server that is configured to allow many user processes to share very few server processes, so the number of users that can be supported is increased. With shared server configuration, many user processes connect to a dispatcher. The dispatcher directs multiple incoming network session requests to a common queue. An idle shared server process from a shared pool of server processes picks up a request from the queue. This means a small pool of server processes can server a large amount of clients. Contrast with dedicated server.

sibling

An entry that has the same parent as one or more other entries.

Signed Public Key And Challenge (SPKAC)

Signed Public Key And Challenge (SPKAC) is a proprietary protocol used by the Netscape Navigator browser to request certificates.

simple authentication

The process by which the client identifies itself to the server by means of a DN and a password which are not encrypted when sent over the network. In the simple authentication option, the server verifies that the DN and password sent by the client match the DN and password stored in the directory.

Simple Authentication and Security Layer (SASL)

A method for adding authentication support to connection-based protocols. To use this specification, a protocol includes a command for identifying and authenticating a user to a server and for optionally negotiating a security layer for subsequent protocol interactions. The command has a required argument identifying a SASL mechanism.

single key-pair wallet

A **PKCS#12**-format wallet that contains a single user **certificate** and its associated **private key**. The **public key** is imbedded in the certificate.

single sign-off

The process by which you terminate an OracleAS Single Sign-On session and log out of all active partner applications simultaneously. You can do this by logging out of the application that you are working in.

single sign-on (SSO)

A process or system that enables a user to access multiple computer platforms or application systems after being authenticated only once.

single sign-on SDK

Legacy APIs to enable OracleAS Single Sign-On partner applications for single sign-on. The SDK consists of PL/SQL and Java APIs as well as sample code that demonstrates how these APIs are implemented. This SDK is now deprecated and [mod_osso](#) is used instead.

single sign-on server

Program logic that enables users to log in securely to single sign-on applications such as expense reports, mail, and benefits.

SLAPD

Standalone LDAP daemon. An LDAP directory server service that is responsible for most functions of a directory except replication.

slave

See [consumer](#).

smart knowledge reference

A [knowledge reference](#) that is returned when the knowledge reference entry is in the scope of the search. It points the user to the server that stores the requested information.

SOAP

Simple Object Access Protocol (SOAP) is an [XML](#)-based protocol that defines a framework for passing messages between systems over the Internet via HTTP. A SOAP message consists of three parts — an envelope that describes the message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses.

specific administrative area

Administrative areas control:

- Subschema administration
- Access control administration
- Collective attribute administration

A *specific* administrative area controls one of these aspects of administration. A specific administrative area is part of an autonomous administrative area.

SPKAC

See [Signed Public Key And Challenge \(SPKAC\)](#).

sponsor node

In replication, the node that is used to provide initial data to a new node.

SSL

See [Secure Sockets Layer \(SSL\)](#).

stream cipher

Stream ciphers are a type of [symmetric algorithm](#). A stream cipher encrypts in small units, often a bit or a byte at a time, and implements some form of feedback

mechanism so that the key is constantly changing. **RC4** is an example of a stream cipher.

See also: **block cipher**.

subACLSubentry

A specific type of **subentry** that contains **access control list (ACL)** information.

subclass

An object class derived from another object class. The object class from which it is derived is called its **superclass**.

subentry

A type of entry containing information applicable to a group of entries in a subtree. The information can be of these types:

- Access control policy points
- Schema rules
- Collective attributes

Subentries are located immediately below the root of an administrative area.

subordinate CA

In a hierarchical **public key infrastructure (PKI)**, the subordinate **Certificate Authority (CA)** is a CA whose certificate signature key is certified by another CA, and whose activities are constrained by that other CA.

subordinate reference

A **knowledge reference** pointing downward in the **directory information tree (DIT)** to a **naming context** that starts immediately below an entry

subschema DN

The list of **directory information tree (DIT)** areas having independent **schema** definitions.

subSchemaSubentry

A specific type of **subentry** containing **schema** information.

subtree

A section of a directory hierarchy, which is also called a **directory information tree (DIT)**. The subtree typically starts at a particular directory node and includes all subdirectories and objects below that node in the directory hierarchy.

subtype

An attribute with one or more options, in contrast to that same attribute without the options. For example, a `commonName (cn)` attribute with American English as an option is a subtype of the `commonName (cn)` attribute without that option. Conversely, the `commonName (cn)` attribute without an option is the **supertype** of the same attribute with an option.

success URL

When using Oracle Application Server Single Sign-On, the URL to the routine responsible for establishing the session and session cookies for an application.

super user

A special directory administrator who typically has full access to directory information.

superclass

The **object class** from which another object class is derived. For example, the object class `person` is the superclass of the object class `organizationalPerson`. The latter, namely, `organizationalPerson`, is a **subclass** of `person` and inherits the attributes contained in `person`.

superior reference

A **knowledge reference** pointing upward to a **directory system agent (DSA)** that holds a naming context higher in the **directory information tree (DIT)** than all the naming contexts held by the referencing DSA.

supertype

An attribute without options, in contrast to the same attribute with one or more options. For example, the `commonName (cn)` attribute without an option is the supertype of the same attribute with an option. Conversely, a `commonName (cn)` attribute with American English as an option is a **subtype** of the `commonName (cn)` attribute without that option.

supplier

In replication, the server that holds the master copy of the **naming context**. It supplies updates from the master copy to the **consumer** server.

symmetric algorithm

A symmetric algorithm is a cryptographic algorithm that uses the same key for encryption and decryption. There are essentially two types of symmetric (or secret key) algorithms — **stream ciphers** and **block ciphers**.

symmetric cryptography

Symmetric cryptography (or shared secret cryptography) systems use the same key to encipher and decipher data. The problem with symmetric cryptography is ensuring a secure method by which the sender and recipient can agree on the secret key. If a third party were to intercept the secret key in transit, they could then use it to decipher anything it was used to encipher. Symmetric cryptography is usually faster than asymmetric cryptography, and is often used when large quantities of data need to be exchanged. **DES**, **RC2**, and **RC4** are examples of symmetric cryptography algorithms.

symmetric key

See **secret key**.

System Global Area (SGA)

A group of shared memory structures that contain data and control information for one Oracle database instance. If multiple users are concurrently connected to the same instance, the data in the instance SGA is shared among the users. Consequently, the SGA is sometimes referred to as the "shared global area." The combination of the background processes and memory buffers is called an Oracle instance.

system operational attribute

An attribute holding information that pertains to the operation of the directory itself. Some operational information is specified by the directory to control the server, for

example, the time stamp for an entry. Other operational information, such as access information, is defined by administrators and is used by the directory program in its processing.

think time

The time the user is not engaged in actual use of the processor.

third-party access management system

Non-Oracle single sign-on system that can be modified to use OracleAS Single Sign-On to gain access to Oracle Application Server applications.

throughput

The number of requests processed by Oracle Internet Directory for each unit of time. This is typically represented as "operations per second."

Time Stamp Protocol (TSP)

Time Stamp Protocol (TSP), as specified in RFC 3161, defines the participating entities, the message formats, and the transport protocol involved in time stamping a digital message. In a TSP system, a trusted third-party Time Stamp Authority (TSA) issues time stamps for messages.

TLS

See [Transport Layer Security \(TLS\)](#).

Transport Layer Security (TLS)

A protocol providing communications privacy over the Internet. The protocol enables client/server applications to communicate in a way that prevents eavesdropping, tampering, or message forgery.

Triple Data Encryption Standard (3DES)

Triple Data Encryption Standard (3DES) is based on the [Data Encryption Standard \(DES\)](#) algorithm developed by IBM in 1974, and was adopted as a national standard in 1977. 3DES uses three 64-bit long keys (overall key length is 192 bits, although actual key length is 56 bits). Data is encrypted with the first key, decrypted with the second key, and finally encrypted again with the third key. This makes 3DES three times slower than standard DES but also three times more secure.

trusted certificate

A third party identity that is qualified with a level of trust. The trust is used when an identity is being validated as the entity it claims to be. Typically, trusted certificates come from a [Certificate Authority \(CA\)](#) you trust to issue user certificates.

trustpoint

See [trusted certificate](#).

TSP

See [Time Stamp Protocol \(TSP\)](#).

Unicode

A type of universal character set, a collection of 64K characters encoded in a 16-bit space. It encodes nearly every character in just about every existing character set standard, covering most written scripts used in the world. It is owned and defined by Unicode Inc. Unicode is canonical encoding which means its value can be passed

around in different locales. But it does not guarantee a round-trip conversion between it and every Oracle character set without information loss.

UNIX Crypt

The UNIX encryption algorithm.

URI

Uniform Resource Identifier (URI). A way to identify any point of content on the Web, whether it be a page of text, a video or sound clip, a still or animated image, or a program. The most common form of URI is the Web page address, which is a particular form or subset of URI called a **URL**.

URL

Uniform Resource Locator (URL). The address of a file accessible on the Internet. The file can be a text file, HTML page, image file, a program, or any other file supported by HTTP. The URL contains the name of the protocol required to access the resource, a domain name that identifies a specific computer on the Internet, and a hierarchical description of the file location on the computer.

URLC token

The OracleAS Single Sign-On code that passes authenticated user information to the **partner application**. The partner application uses this information to construct the session cookie.

user name mapping module

A OracleAS Single Sign-On Java module that maps a user **certificate** to the user's nickname. The nickname is then passed to an authentication module, which uses this nickname to retrieve the user's certificate from the directory.

user search base

In the Oracle Internet Directory default **directory information tree (DIT)**, the node in the identity management realm under which all the users are placed.

UTC (Coordinated Universal Time)

The standard time common to every place in the world. Formerly and still widely called Greenwich Mean Time (GMT) and also World Time, UTC nominally reflects the mean solar time along the Earth's prime meridian. UTC is indicated by a z at the end of the value, for example, 200011281010z.

UTF-8

A variable-width 8-bit encoding of **Unicode** that uses sequences of 1, 2, 3, or 4 bytes for each character. Characters from 0-127 (the 7-bit ASCII characters) are encoded with one byte, characters from 128-2047 require two bytes, characters from 2048-65535 require three bytes, and characters beyond 65535 require four bytes. The Oracle character set name for this is AL32UTF8 (for the Unicode 3.1 standard).

UTF-16

16-bit encoding of **Unicode**. The Latin-1 characters are the first 256 code points in this standard.

verification

Verification is the process of ensuring that a given **digital signature** is valid, given the **public key** that corresponds to the **private key** purported to create the signature and the data block to which the signature purportedly applies.

virtual host

A single physical Web server machine that is hosting one or more Web sites or domains, or a server that is acting as a proxy to other machines (accepts incoming requests and reroutes them to the appropriate server).

In the case of OracleAS Single Sign-On, virtual hosts are used for load balancing between two or more OracleAS Single Sign-On servers. They also provide an extra layer of security.

virtual host name

In an Oracle Application Server Cold Failover Cluster (Identity Management), the host name corresponding to a particular virtual IP address.

virtual IP address

In an Oracle Application Server Cold Failover Cluster (Identity Management), each physical node has its own physical IP address and physical host name. To present a single system image to the outside world, the cluster uses a dynamic IP address that can be moved to any physical node in the cluster. This is called the virtual IP address.

wait time

The time between the submission of the request and initiation of the response.

wallet

An abstraction used to store and manage security credentials for an individual entity. It implements the storage and retrieval of credentials for use with various cryptographic services. A wallet resource locator (WRL) provides all the necessary information to locate the wallet.

Wallet Manager

See [Oracle Wallet Manager](#).

Web service

A Web service is application or business logic that is accessible using standard Internet protocols, such as [HTTP](#), [XML](#), and [SOAP](#). Web Services combine the best aspects of component-based development and the World Wide Web. Like components, Web Services represent black-box functionality that can be used and reused without regard to how the service is implemented.

Web Services Description Language (WSDL)

Web Services Description Language (WSDL) is the standard format for describing a Web service using [XML](#). A WSDL definition describes how to access a Web service and what operations it will perform.

WSDL

See [Web Services Description Language \(WSDL\)](#).

WS-Federation

Web Services Federation Language (WS-Federation) is a specification developed by Microsoft, IBM, BEA, VeriSign, and RSA Security. It defines mechanisms to allow [federation](#) between entities using different or like mechanisms by allowing and brokering trust of identities, attributes, and authentication between participating [Web services](#).

See also: [Liberty Alliance](#).

X.500

X.500 is a standard from the International Telecommunication Union (ITU) that defines how global directories should be structured. X.500 directories are hierarchical with different levels for each category of information, such as country, state, and city.

X.509

X.509 is the most widely used standard for defining digital certificates. A standard from the International Telecommunication Union (ITU), for hierarchical directories with authentication services, used in many **public key infrastructure (PKI)** implementations.

XML

Extensible Markup Language (XML) is a specification developed by the World Wide Web Consortium (W3C). XML is a pared-down version of Standard Generalized Mark-Up Language (SGML), designed especially for Web documents. XML is a metalanguage (a way to define tag sets) that allows developers to define their own customized markup language for many classes of documents.

XML canonicalization (C14N)

This is a process by which two logically equivalent XML documents can be resolved to the same physical representation. This has significance for digital signatures because a signature can only verify against the same physical representation of the data against which it was originally computed. For more information, see the W3C's XML Canonicalization specification.

A

abandoning an operation, 11-29
access control, 2-4, 2-5
 and authorization, 2-5
access control information (ACI), 2-6
 attributes, 2-5
 directives
 format, 2-6
Access Control List (ACL), 2-5
access control lists (ACLs), 2-5
ACI. See access control information (ACI)
ACLs. See Access Control List (ACL)
anonymous authentication, 2-5
application login, 9-12
application logout, 9-12
application session cookie
 clearing, 9-10
 coding for, 9-10
applications, building
 with the C API, 11-44
attributes
 types, 2-3
 values, 2-3
authentication, 2-4
 anonymous, 2-5
 certificate-based, 2-5
 modes, SSL, 11-1, 11-2
 one-way SSL, 2-5
 options, 2-4
 password-based, 2-5
 SSL, 2-5, 11-1
 none, 11-2
 one-way, 11-2
 two-way, 11-2
 strong, 2-5
 to a directory server
 enabling, 2-10
 enabling, by using DBMS_LDAP, 2-11
 enabling, by using the C API, 2-10
 to the directory, 11-10
 two-way SSL, 2-5
authentication, simple, 9-6
authorization, 2-4, 2-5
authorization ID, 2-4

B

bulk tools, 1-10

C

C API
 functions
 abandon, 11-29
 abandon_ext, 11-29
 add, 11-25
 add_ext_s, 11-25
 add_s, 11-25
 compare, 11-20
 compare_ext, 11-20
 compare_ext_s, 11-20
 compare_s, 11-20
 count_entries, 11-35
 count_references, 11-35
 count_values, 11-37
 count_values_len, 11-37
 delete, 11-26
 delete_ext, 11-26
 delete_ext_s, 11-26
 delete_s, 11-26
 dn2ufn, 11-38
 err2string, 11-32
 explode_dn, 11-38
 explode_rdn, 11-38
 extended_operation, 11-28
 extended_operation_s, 11-28
 first_attribute, 11-36
 first_entry, 11-35
 first_message, 11-34
 first_reference, 11-35
 get_dn, 11-38
 get_entry_controls, 11-39
 get_option, 11-6
 get_values, 11-37
 get_values_len, 11-37
 init_ssl call, 11-2
 modify, 11-21
 modify_ext, 11-21
 modify_ext_s, 11-21
 modify_s, 11-21
 msgid, 11-30

- msgtype, 11-30
- next_attribute, 11-36
- next_entry, 11-35
- next_message, 11-34
- next_reference, 11-35
- parse_extended_result, 11-32
- parse_reference, 11-39
- parse_result, 11-32
- parse_sasl_bind_result, 11-32
- rename, 11-23
- rename_s, 11-23
- result, 11-30
- sasl_bind, 11-10
- sasl_bind_s, 11-10
- search_st, 11-17
- set_option, 11-6
- simple_bind, 11-10
- simple_bind_s, 11-10
- unbind_ext, 11-16
- unbind_s, 11-16
- value_free, 11-37
- value_free_len, 11-37
- sample usage, 11-40
- summary, 11-3
- usage with SSL, 11-40
- usage without SSL, 11-41
- certificate authority, 2-5
- certificate-based authentication, 2-5
- certificates, 2-5
- children of an entry, listing, 11-20
- code examples
 - application login, 9-12
 - authentication, 9-6, 9-7
 - forced authentication, 9-8, 9-12
 - single sign-off, 9-8
- components
 - Oracle Internet Directory SDK, 1-4
- controls, working with, 3-10, 3-12, 11-14

D

- DAP Information Model, 2-3
- DAS units, 8-1
- DAS URL Parameter Descriptions, 15-5
- DAS URL Parameters, 8-3
- DAS URL parameters, 15-2
- data
 - integrity, 2-4, 2-6
 - privacy, 2-4, 2-6
- data-type summary, 12-5
- DBMS_LDAP package
 - searching by using, 2-11
- DBMS_LDAP_UTL
 - about, 14-1
 - data-types, 14-34
 - function return codes, 14-32
 - group-related subprograms
 - about, 14-2
 - function create_group_handle, 14-14
 - function get_group_dn, 14-17

- function get_group_properties, 14-16
- function set_group_handle_properties, 14-15
- miscellaneous subprograms
 - about, 14-2
 - function check_interface_version, 14-30
 - function create_mod_propertyset, 14-28
 - function get_property_names, 14-24
 - function get_property_values, 14-25
 - function get_property_values_len, 14-26
 - function normalize_dn_with_case, 14-24
 - function populate_mod_propertyset, 14-29
 - procedure free_handle, 14-30
 - procedure free_mod_propertyset, 14-29
 - procedure free_propertyset_collection, 14-27
- subscriber-related subprograms
 - about, 14-2
 - function create_subscriber_handle, 14-19
 - function get_subscriber_dn, 14-21
 - function get_subscriber_properties, 14-19
- user-related subprograms
 - about, 14-1
 - function authenticate_user, 14-3
 - function check_group_membership, 14-11
 - function create_user_handle, 14-5
 - function get_group_membership, 14-13
 - function get_user_dn, 14-10
 - function get_user_extended_properties, 14-9
 - function get_user_properties, 14-6
 - function locate_subscriber_for_user, 14-12
 - function set_user_handle_properties, 14-5
 - function set_user_properties, 14-7

dependencies and limitations, 11-45

C API, 11-45

DES40 encryption, 2-6

directives, 2-6

Directory Information Tree, 2-2

directory information tree (DIT), 2-2

directory server discovery, 3-4

distinguished names, 2-2

components of, 2-2

format, 2-2

DNs. see distinguished names.

documentation, related, 0-xxii

dynamic directives

common types, 9-3

defined, 9-2, 9-3

programming languages supported, 9-3

dynamic password verifiers

controls, 3-10, 3-12

creating, 3-10 to 3-12

parameters, 3-10, 3-11

E

encryption

DES40, 2-6

levels available in Oracle Internet Directory, 2-6

RC4_40, 2-6

entries

distinguished names of, 2-2

- locating by using distinguished names
- naming, 2-2
- reading, 11-19

errors

- handling and parsing results, 11-32

exception summary, 12-3

F

- filters, 2-14
- forced authentication, 9-8, 9-12
- formats, of distinguished names, 2-2

G

- GET authentication method, 9-9
- global user inactivity timeout, 9-9

H

- header files and libraries, required, 11-44
- history of LDAP, 2-1
- HTTP headers, 9-1

I

- integrity, data, 2-6
- interface calls, SSL, 11-2

J

- J2EE security APIs, 10-1
- JAAS policy management APIs, 10-4
- Java, 1-4, 2-8
- Java API reference
 - class descriptions
 - Property class, 4-2
 - PropertySet class, 4-2
 - PropertySetCollection class, 4-2
- Java APIs for Oracle Internet Directory, 13-1
- Java partner applications
 - dynamically protected, 9-6, 9-9
 - statically protected, 9-6
- Java partner applications, statically protected, 9-5
- JAZN
 - see Oracle Application Server Java Authentication and Authorization Service
- JNDI, 1-4, 2-8
- JNDI location, 13-1

L

LDAP

- functional model, 2-3
- history, 2-1
- information model, 2-3
- messages, obtaining results and peeking
 - inside, 11-30
- naming model, 2-2
- operations, performing, 11-16
- security model, 2-4

- session handle options, 11-6
 - in the C API, 2-10
- sessions
 - initializing, 2-8
 - version 2 C API, 11-1
- LDAP APIs, 1-6
- LDAP Functional Model, 2-3
- LDAP Models, 2-2
 - LDAP Naming Model, 2-2
- LDAP Security Model, 2-4
- ldapadd
 - plug-in support, 7-20 to 7-22
- ldap-bind operation, 2-4
- ldapcompare
 - plug-in support, 7-22 to 7-25
- ldapmodify
 - plug-in support, 7-18 to 7-20

M

mod_osso

- benefits, 9-1
- compared with single sign-on SDK, 9-1
- definition, 9-1
- integration methods, 9-2
- sample applications, 9-3, 9-9

mod_osso cookie, 9-10

N

- naming entries, 2-2

O

- OC4J security APIs, 10-2
- one-way SSL authentication, 2-5, 11-2
- OpenLDAP Community, 0-xxii
- operational attributes
 - ACI, 2-5
- Oracle Application Server Java Authentication and Authorization Service
 - defined, 1-2
- Oracle Directory Manager, 1-9
- Oracle directory replication server, 1-9
- Oracle directory server, 1-9
- Oracle extensions
 - application
 - deinstallation logic, 1-6
 - runtime logic, 1-6
 - shutdown logic, 1-6
 - startup and bootstrap logic, 1-5
 - group management functionality, 3-3
 - programming abstractions
 - for Java language, 4-1, 5-1
 - for PL/SQL language, 5-1
 - programming abstractions for Java language, 4-1, 5-1
 - user management functionality, 4-1, 5-1
- Oracle extensions to support SSL, 11-1
- Oracle Identity Management
 - infrastructure

- modifying existing applications, 1-2
- integrating
 - new applications, 1-3
- integrating applications with, 1-1
 - benefits of, 1-1
 - supported services, 1-2
- Oracle Internet Directory, components, 1-9
- Oracle SSL call interface, 11-1
- Oracle SSL extensions, 11-1
- Oracle SSL-related libraries, 11-45
- Oracle system libraries, 11-45
- Oracle wallet, 11-2
- Oracle Wallet Manager, 11-2
 - required for creating wallets, 11-45
- Oracle xxtensions
 - what an LDAP-integrated application looks like, 1-4
- OracleAS Single Sign-On
 - user attributes, 9-1
- overview of LDAP models, 2-2

P

- password-based authentication, 2-5
- passwords
 - policies, 2-6
- permissions, 2-4, 2-5
- PL/SQL API, 12-1
 - contains subset of C API, 2-8
 - data-type summary, 12-5
 - exception summary, 12-3
 - functions
 - add_s, 12-30
 - ber_free, 12-37
 - bind_s, 12-7
 - compare_s, 12-9
 - count_entries, 12-15
 - count_values, 12-32
 - count_values_len, 12-32
 - create_mod_array, 12-24
 - dbms_ldap.init, 12-6
 - delete_s, 12-21
 - err2string, 12-23
 - explode_dn, 12-34
 - first_attribute, 12-16
 - first_entry, 12-13
 - get_dn, 12-18
 - get_values, 12-19
 - get_values_len, 12-20
 - init, 12-5
 - modify_s, 12-29
 - modrdn2_s, 12-22
 - msgfree, 12-36
 - next_attribute, 12-17
 - next_entry, 12-14
 - open_ssl, 12-35, 12-36, 12-37
 - rename_s, 12-33
 - search_s, 12-10
 - search_st, 12-12
 - simple_bind_s, 12-6

- unbind_s, 12-8
- loading into database, 2-8
- procedures
 - free_mod_array, 12-31
 - populate_mod_array (binary version), 12-25
 - populate_mod_array (string version), 12-25
 - subprograms, 12-5
 - summary, 12-1
- plug-ins
 - binary support, 7-18 to 7-25
 - provisioning interface, A-1
- policy management APIs, 10-4
- POST authentication method, 9-9
- privacy, data, 2-4, 2-6
- privileges, 2-4, 2-5
- procedures, PL/SQL
 - free_mod_array, 12-31
 - populate_mod_array (binary version), 12-25
 - populate_mod_array (string version), 12-25
- provisioning interface plug-ins, A-1

R

- RC4_40 encryption, 2-6
- RDNs. see relative distinguished names (RDNs)
- related documentation, 0-xxii
- relative distinguished names (RDNs), 2-2
- results, stepping through a list of, 11-34
- RFC 1823, 11-45

S

- sample C API usage, 11-40
- SDK components, 1-4
- search
 - results
 - parsing, 11-35
 - scope, 2-13
 - search-related operations, flow of, 2-12
 - security APIs, 10-1, 10-2
 - security, within Oracle Internet Directory environment, 2-4
 - self-service console, 8-2
 - service location record, 3-4
- servlets
 - dynamically protected, 9-6, 9-9
 - statically protected, 9-5, 9-6
- sessions
 - closing, 11-16
 - enabling termination by using DBMS_LDAP, 2-17
 - initializing
 - by using DBMS_LDAP, 2-9
 - by using the C API, 2-8
 - session-specific user identity, 2-4
- simple authentication, 2-5
- single sign-off, 9-8
- single sign-on SDK
 - compared with mod_osso, 9-1
- Smith, Mark, 0-xxii

SSL

- authentication modes, 11-1
 - default port, 2-5
 - handshake, 11-2
 - interface calls, 11-2
 - no authentication, 2-5
 - one-way authentication, 2-5
 - Oracle extensions, 11-1
 - provide encryption and decryption, 11-1
 - two-way authentication, 2-5
 - wallets, 11-2
- static directives
- defined, 9-2
 - writing, 9-2
- strong authentication, 2-5

T

- TCP/IP socket library, 11-45
- two-way authentication, SSL, 11-2
- types of attributes, 2-3

U

- URLs, protecting, 9-2, 9-3
- user attributes, 9-1, 9-2

W

- wallets
- SSL, 11-2
 - support, 11-2

